Hindawi
International Journal of Reconfigurable Computing
Volume 2021, Article ID 5584391, 31 pages
https://doi.org/10.1155/2021/5584391

Research Article

A Service-Oriented Component-Based Framework for Dynamic Reconfiguration Modeling Targeting SystemC/TLM

Khaled Allem,[1] **El-Bay Bourennane**,[2] **and Youcef Khelfaoui**[3]

[1]*LIMED Laboratory, University of Bejaia, Bejaia 06000, Algeria*
[2]*LE2I Laboratory, University of Burgundy, Dijon, France*
[3]*L2ME Laboratory, University of Bejaia, Bejaia 06000, Algeria*

Correspondence should be addressed to Khaled Allem; khaled.allem@univ-bejaia.dz

Received 20 February 2021; Revised 6 June 2021; Accepted 19 July 2021; Published 3 August 2021

Academic Editor: João Cardoso

To deal with the complex design issues of Dynamically Reconfigurable Systems-on-Chip (DRSoCs), it is extremely relevant to raise the abstraction level in which models are expressed. A high abstraction level allows great flexibility and reusability while bypassing low-level implementation details. In this context, model-driven engineering (MDE) provides support to build and transform precise and structured models for a particular purpose at different levels of abstraction. Indeed, high-level models are successively refined to low-level models until reaching the executable ones. Thus, this paper presents an MDE-based framework for DRSoCs design enabling the transformation of UML/MARTE specifications to SystemC/TLM implementation. To achieve a high degree of expressiveness for modeling dynamic reconfiguration, we use a suitable software engineering approach based on service-oriented component architecture. Since MARTE does not cover the common features of dynamic reconfiguration domain and service orientation concepts, new stereotypes are created by refinement to add missing capabilities to the profile. Likewise, SystemC does not provide native support for dynamic reconfiguration, thus leading us to adopt a design pattern based solution for DRSoCs implementation in compliance with standards. The proposed framework is validated through a reconfigurable active 3-way crossover case study in which we demonstrate the practicability of the approach by gradual model transformations with reduced implementation effort and significant design productivity gain.

1. Introduction

Raising the abstraction level in order to overcome the explosive complexity and competitive pressures of Systems-on-Chip (SoCs) especially those qualified as dynamically reconfigurable is highly recommended by designers. Electronic system-level methodologies (ESL) enable the use of appropriate abstractions to achieve as quickly as possible simulation models in order to avoid time-consuming low-level simulations. For ESL design, electronic design automation (EDA) tools have been developed for the specification, design, verification, implementation, and test of electronic systems. To address challenges of ESL tasks, transaction-level modeling (TLM) has emerged as an efficient methodology with an acceptable simulation speed and modeling accuracy compared to register transfer level (RTL) [1]. Of all system-level description languages, SystemC [2] seems to be the most appropriate to meet the TLM requirements. SystemC-based transaction-level modeling involves communication between SystemC processes using function calls, while respecting the principle of separating communication from computation. By replacing all pin-level events with a single function call, it is possible to reach speedup factors up to 10.000 x [3].

Dynamically Reconfigurable Systems (DRSs) can be defined differently depending on the research community. In electronics, DRSs are able to dynamically modify their functions at runtime allowing the activation (addition) and deactivation (removal) of hardware components [4]. A DRS is considered in [5] as a system whose subsystems can be modified or have their settings modified during operation to achieve a specific goal. In software engineering, adaptive

systems assimilated to DRS are subject to behavior or structure adaptation at runtime in response to unplanned or planned execution changes (user requirements, resources limitations, and environmental conditions) [6]. Most reconfigurable systems are based on FPGA (field-programmable gate array) technology; their dynamic partial reconfiguration (DPR) capability enables reconfiguring parts of them during runtime without the need to stop the device. Such systems, capable of dynamically loading and unloading IP Cores (Intellectual Property Cores) while the rest of the system is still running, are called Dynamically Reconfigurable Systems-on-Chip (DRSoCs). For modeling DRSoC features at a high abstraction level, traditional codesign techniques become obsolete; indeed, vendor design flows such as ISE or Vivado design tools from Xilinx work at a relatively low abstraction level [7]. Although there is a plethora of methodologies for embedded system codesign, the model-driven engineering (MDE) seems to federate both research and industry communities by supporting abstract models and automated methods to process them. These models may be expressed with unified modeling language (UML) [8] or with domain-specific languages (DSL) like AutoModel [9], D'Artagnan [10], ESPeciaL [11] or with UML profiles such as TUT [12], UML for SystemC [13–15], SysML [16], and UML profile for modeling and analysis of real-time embedded systems (MARTE) [17]. Concretely, none of these languages and profiles has been developed to cover comprehensively DRSoC features; we are faced with the major challenge of filling the lack of UML-based tools for DPR modeling.

When trying to model dynamic reconfiguration in SystemC, the designer has to circumvent a significant hurdle; indeed, new instances cannot be generated once simulation has already started prohibiting any dynamic creation of entities. This problem would be wrongly tackled by suspending the simulation and making changes to the elaboration phase. The elaboration aims to create internal data structures within the kernel as required to support the semantics of simulation; thus, the dynamic reconfiguration process is still not modeled [18]. Without altering the SystemC kernel while complying with the standard, two different strategies for DPR modeling may be used to represent reconfigurable areas: treating them as C++ objects or as SystemC modules [19]. Modeling reconfigurable areas as C++ objects makes it impossible to use SystemC dynamic processes, given that objects are passive. For a hardware designer, it is more intuitive to use modules than to use objects.

In order to get a high expressiveness degree for DPR modeling, we explored software engineering approaches for dynamic adaptation. An approach that fulfills DPR requirements would be to conciliate both service-oriented and component-based approaches resulting in service component architecture (SCA) [20–22]. Indeed, the two approaches are complementary; a component is an implementation that provides and requires resources while a service is a runtime artifact that publishes an interface and that requests dynamically the service providers it needs [21]. To leverage SCA extension breakthroughs for software development, we applied its principles to DRSoC modeling especially since hardware design is naturally component-based. Two main advantages of integrating SCA concepts into SoC platform are worth highlighting. Firstly, heterogeneous interoperable components can be seamlessly integrated at different abstraction levels resulting in an effective design space exploration. Secondly, interchangeable modules communicate through well-defined interfaces without too many implementation details, thereby improving the scalability and substitutability in DRSoCs beyond simply reusing IP [23].

To cope with the lack of partial and dynamic reconfigurable FPGA modeling tools at ESL, we propose a MARTE-based service-oriented component modeling framework targeting the automatic generation of SystemC code at transaction level. Key contributions are as follows:

(i) A high-level design flow is provided to enable SystemC/TLM code generation from an extended MARTE front end according to the model-based development tenets. A fully integrated Eclipse toolchain is built to materialize the proposal.

(ii) The MARTE profile is extended with DPR and SCA domain-specific semantics for systems modeling, since no specified support is provided by the MARTE standard for this purpose.

(iii) A reconfigurable SystemC module for dynamic spawned creation is modeled and implemented relying on calls to sc_spawn function and use of the abstract factory design pattern providing the necessary level of abstraction.

(iv) The MARTE profile is also extended with SystemC/TLM concepts in order to smoothly move to the executable code of the underlying system.

The remainder of the paper is organized as follows. Section 2 gives a brief overview of a set of modeling concepts, languages, and tools used in this work. Section 3 presents the previous literature related to DPR modeling at a high abstraction level. Following a bottom-up approach (driven from the SystemC platform), it reviews the most important solutions for modeling Dynamically Reconfigurable Systems with SystemC and UML. Section 4 details the proposed modeling framework with an emphasis on MARTE profile specialization covering DPR specific concepts while describing the refined stereotypes of the involved subprofiles. Section 5 experimentally validates the framework and projects our proposal on a reconfigurable audio FIR filter within an active 3-way crossover. Section 6 closes this paper and presents further perspectives for future work. Details on transformation rules, generated SystemC/TLM model, and automatic code generation of the business logic are described in the appendix.

For the sake of clarity, Table 1 lists the abbreviations used throughout this paper.

2. Preliminaries

In this section, we will introduce some preliminary knowledge for understanding the work.

Metamodel. The abstract syntax of UML is specified using a UML model called the UML metamodel. The

Table 1: List of abbreviations used in this research article.

Abbreviation	Meaning
ATL	Atlas Transformation Language
DPR	Dynamic partial reconfiguration
DRS	Dynamically Reconfigurable System
DRSoC	Dynamically Reconfigurable System-on-Chip
EMF	Eclipse Modeling Framework
ESL	Electronic system-level
FIR	Finite impulse response
FPGA	Field-programmable gate array
GCM	Generic component model
GQAM	Generic quantitative analysis modeling
GRM	Generic resource modeling
HLAM	High-level application modeling
HRM	Hardware resource modeling
IP	Intellectual property
MARTE	Modeling and analysis of real-time and embedded systems
MDE	Model-driven engineering
NFP	Nonfunctional properties
RTL	Register transfer level
SCA	Service component architecture
SoC	System-on-chip
SOCM	Service-oriented component model
SRM	Software resource modeling
TLM	Transaction-level modeling
UML	Unified modeling language

abstract syntax defines the set of UML modeling concepts, their attributes, and their relationships, as well as the rules for combining these concepts to construct UML models [8]. The UML metamodel is MOF-based and typically consists of metaclasses and meta-associations with their properties.

Stereotype. A stereotype defines an extension for one or more metaclasses and allows the use of platform or domain-specific terminology or notation in place of, or in addition to, those used for the extended metaclasses [8].

Extension. An extension is a kind of association used to indicate that the properties of a metaclass are extended through a stereotype and gives the ability to flexibly add/remove stereotypes to/from classes.

Profile. A profile defines limited extensions to a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain. The primary extension construct is the stereotype. It is possible for one profile to reuse all of or parts of another and to extend other profiles.

MARTE. The UML profile for modeling and analysis of real-time and embedded systems defines foundations for model-based descriptions of real-time and embedded systems covering the entire development process. It is composed of extension units (subprofiles and model libraries) each involved in several use cases. In the context of MDE, MARTE defines precise semantics facilitating the automatic transformation of models as well as the integration of analysis and modeling tools.

SystemC. SystemC is a C++ class library providing a mechanism for modeling hardware and software together at multiple levels of abstraction. Systems are represented by a module hierarchy to manage structure and connectivity. A module can contain ports, exports, channels, processes, events, instances of other modules, other data members, and member functions. SystemC is often associated with electronic system-level (ESL) design and with transaction-level modeling (TLM).

TLM. TLM is a transaction-based modeling approach for describing systems at a higher abstraction level above RTL, with emphasis on the separation of communication from computation within a system. By eliminating unnecessary details, it offers an adequate trade-off between simulation speed and accuracy while enabling interoperability of models. TLM-2.0 classes are layered on top of the SystemC class library and consist of a set of core interfaces, the global quantum, the initiator and target sockets, the generic payload and base protocol, and the utilities.

sc_spawn. sc_spawn is a SystemC function used to create a static or dynamic spawned process instance. A spawned process is typically a dynamic process, but if sc_spawn is called before the end of elaboration, it would be a static process. Function sc_spawn may be called during elaboration or from a static, dynamic, spawned, or unspawned process during the simulation phase [2].

Papyrus. Papyrus is an environment for editing any type of Eclipse Modeling Framework (EMF) model, in particular supporting UML2 and associated modeling languages such as SysML and MARTE. Papyrus is a complete UML modeling environment, it can also be used to develop UML profiles and generate code from UML models.

ATL. ATL (Atlas Transformation Language) is a model transformation language specified as both a metamodel and a textual concrete syntax. In MDE, ATL allows producing a number of target models from a set of source models through transformation rules.

Acceleo. Acceleo is a template-based code generator that uses any EMF based models to generate any kind of code according to the principles of model-to-text transformation.

3. Related Works

The modeling of embedded systems with UML has been the subject of several research works [9, 24–27]. However, few of them deal with DPR features at high abstraction level. Existing works fall short of sufficiently addressing both elevation of design abstraction levels and precise DPR semantics inside modeling tools. While the majority of works advocate the use of UML and/or SystemC for DPR modeling, others focus on optimization at RTL. Furthermore, two main branches dominate research on DRS modeling with SystemC. The first one proposes solutions based on modified SystemC kernel [28, 29], while the second promotes compliance with the standard without altering the simulation kernel. That being said, most researchers recommend complying with the standard and consider non-standard simulation kernel as a bottleneck for design process.

In [30], the proposed approach involves a Process-based Reconfigurable SystemC Module (PRM) for design space exploration speedup. The PRM uses Unix calls for process

manipulation and interprocess communication leading to an OS-dependent solution. A PRM is composed of a static part, a dynamic part, and a user control server (single-request Unix server). Partial dynamic reconfiguration is modeled by assigning different processes to the PRM in a static way and selecting one of them each time, therefore limiting the addition of new configurations during simulation.

The authors in [31] propose a layered dynamic resource manager that provides a set of services to perform efficient scheduling and control of the sequence and processes related to the partial reconfiguration of FPGAs. Furthermore, a SystemC-based simulation framework has been defined by introducing the concept of Dynamic Module. The reconfiguration process is simulated as swap between the different module models running in different SystemC threads by switching from one to another inside the Dynamic Module. The main drawback of this framework is that, for a Dynamic Module, the number of threads implementing its various behaviors must be determined statically at compile time.

Many approaches for modeling reconfigurable SoCs and NoCs based on ASIC with SystemC standard features and TLM at both system level and RTL are presented in [32]. Special types of socket are used to model the selectors concept allowing connection of different lines to the same port at different times. The dynamically reconfigured functional blocks are modeled with reconfigurable components based on a look-up table. Though the set of currently used connections is determined dynamically during simulation, all components are generated statically and possible connections between components are statically implemented. The dynamic reconfiguration of a component can be implemented by using reconfigurable library elements. A reconfigurable element is associated with a class, and the behavior of a specific instance of a class can be dynamically configured by using C++ case operator. As a second solution, the reconfiguration possibilities are specified in terms of logic functions. Then, sc_spawn may be used to implement a dynamic child process with the required functionality. No details are given concerning the components reusability and interoperability, especially as the dynamic reconfiguration possibilities for ASIC are significantly limited compared to FPGA.

For an effective multilevel simulation, a SystemC-based refinement flow of A-HetSC Adaptive Processes (HAPs) into OSSS + R description is presented in [33]. The core of a HAP pattern is a SystemC process which computes adaptive functionality received as a function pointer from its environment which will be substituted by the selection among a finite set of adaptive functionalities statically fixed from an OSSS + R description. Next, the resulting untimed mode-based HAP is refined to a clocked synchronous mode-based HAP, and the adaptive functionality is still included within a SC_THREAD process. The reconfigurable object wrapping consists in converting the mode functions within the context of the adaptive module into implementation classes. In the last refinement step, the asynchronous threads are replaced by SC_CTHREADs to work with bounded loops and to replace dynamic memory handling by code based on static data structures. As in OSSS + R library, the adaptive computations are assigned to HAP modules in a static way, making it impossible to update the set of adaptive functionalities in runtime.

The authors in [34] propose a Y-chart based methodology to model dynamically reconfigurable architectures at high level of abstraction by using SystemC/TLM. On the one hand, they attempt to provide an easy way to develop scheduling strategies for hardware task management, and, on the other hand, they introduce design space exploration in their methodology to provide the developer with a design flow completely integrated in Xilinx partial reconfiguration flow. In the described model, a reconfigurable module is composed of two SystemC dynamic threads: User Algorithm thread spawned during the execution and representing the functionality of the reconfigurable module and Reconfig Control thread which is responsible for the creation and the destruction of the User Algorithm. In addition, a reconfigurable module contains target and initiator sockets used for communication between modules in accordance with the TLM2.0 base protocol. Each component communication interface is associated with a pool of functions called socket control used to connect the sockets to the module. The presented methodology was later used to build the RecoSim reconfigurable simulator integrated into FoRTReSS design flow [35].

ReChannel library [36] extends SystemC with advanced language constructs for high-level reconfiguration modeling. Portals are introduced to connect a channel of the static design part to ports of reconfigurable modules. Portals for all SystemC channel interfaces are provided by the ReChannel library. The dynamic reconfiguration is assimilated to circuit switch allowing the activation of only one module at a time if all its portals can be switched. While reconfigurable modules are created from static ones via a special macro, rc_control object provides registration and reconfiguration control functions for modules. ReChannel also provides a set of language extensions used for explicit description of reconfiguration allowing modules resetting without SystemC kernel altering. However, the tasks are statically assigned to the reconfigurable zones at the beginning of simulation, thereby prohibiting task mobility. A top-down modeling methodology built on top of ReChannel to perform simulation-based functional verification of dynamic reconfigurable systems is proposed in [37]. Related to dynamic reconfiguration challenges, potential bugs have been identified and categorized according to their occurrence before, during, or after reconfiguration while covering the behavioral level, TLM, and RTL.

In the works presented above, the approaches have neither a sufficiently high abstraction level nor the appropriate automation mechanisms necessary to make them more productive from the early stages of development.

In [18], a specific communication adapter is used to provide transparency between components and avoid unnecessary coupling between functionality and low-level integration details by implementing a basic middleware services and using TLM as the physical transport layer for messages. Dynamic libraries combined with a component

adaptor called reconfigurable unit embedding different behaviors are used for modeling DPR relying on C++ plugins and dynamic threads. A reconfiguration controller is used to load dynamically the behaviors associated with each reconfigurable unit and configure the adapter to be accessible from other components in system. A location service is used to map physical and logical addresses responding to the adapter translation requests. Although the separation of logical and physical addressing spaces increases application flexibility, the choice of a particular component always depends on its implementation, which is the major drawback of component-based approaches. These same concepts were taken over by [7] in a full Eclipse integrated design flow of dynamic reconfigurable systems enabling the automatic generation of executable models and FPGA programming files from UML/MARTE high-level models.

The authors in [38] present a MDE-based Gaspard2 framework for implementation of DRSoCs, in which automatic VHDL/C code at RTL is generated from high-level MARTE models. The proposed design flow generates both the code for a dynamically reconfigurable region related to a high-level application model, and control semantics used for the generation of the reconfiguration controller source code. Works [7, 38] use MARTE standard concepts without any specific DPR semantics support, which limits models expressiveness and affects the quality of the generated code.

In [39], a codesign MARTE-based approach is used to model DRS targeting Multiprocessor System on Programmable Chip (MPSoPC). For platform modeling, a reconfigurable zone is specified by the HwPLD stereotype from MARTE HRM profile. At the application level, strategy and state software design patterns are used to model, respectively, the dynamically swap algorithms and the behavior depending state of an object in the application. The adaptive stereotype extending RtUnit is used to model the dynamic component. The reconfiguration method can be strategy-based or state-based which is modeled by reconf_op and reconf_state tagged values, respectively. After allocating the client component into a HwProcessor and the dynamic component into HwPLD, a code targeting Xilinx FPGAs is generated. The limited extension of MARTE described in this work is still insufficient to model the DPR basic concepts.

A MDE-based methodology for DRS codesign is presented in [40]. To model DPR concepts, a MARTE extension called RecoMARTE has been proposed. At the application level, the RtUnit stereotype is extended to ReconfigurableRtUnit to describe reconfigurable tasks. For component interconnection, the ExtendedFlowPort stereotype is used to specify the port kind. At the deployed allocation level, Deployed, IP and CodeFile stereotypes extending, respectively, NamedElement, Class, and Artifact metaclasses are introduced. For modeling physical architecture, the HwRegion stereotype extending HRM HwResource is defined, and from it HwStaticRegion and HwReconfigurableRegion stereotypes are derived. Physical ports are modeled with HwPort stereotype extending HwComponent. The final MARTE model is transformed into IP-XACT intermediate description before generating Xilinx XPS specification files. A RecoMARTE-based framework for fast prototyping of DRSoCs is presented in [41]. While the authors focused on both application and architecture modeling before their mapping, the underlying case study only highlights the use of the Deployed stereotype to describe the allocation model. In another work [42], a RecoMARTE extension is provided to specify a reconfiguration controller for DRSoCs. For constraint verification using NFPs values, an extension of the NFP concepts is proposed. An NfpType has been extended by NfpMeasure stereotype enabling the combination and comparison of values of NfpMeasure types. Additionally, the Controller stereotype extending the RtUnit has been proposed for control definition. From the RecoMARTE models, different transformations are carried out to obtain a controller specification in BZR language ready for discrete controller synthesis.

Despite covering several features of DPR, RecoMARTE defines semantics for Xilinx coding style with advanced design skills, which may alter the high level of abstraction sought.

Table 2 summarizes a comparison between the related works and our proposed framework. For this, we define some useful criteria to determine the expressiveness of high-level DPR models and the effectiveness of the associated approaches. Unlike the very limited extensions proposed in some works, our adaptation is intended to be more complete. Indeed, the proposed extension involves more packages and offers more stereotypes covering the fundamental concepts of DPR. In addition, the adopted service orientation enables a higher level of dynamicity and self-adaptability compared to the component-based paradigm.

4. The Proposed Modeling Framework

In the context of SCA, to explicitly provide dynamic availability support in component models, service-oriented component models (SOCMs) have been introduced. A SOCM merges modularity and separation of concerns of traditional component models with loose coupling, late binding, and runtime services discovery of the service orientation. Loose coupling aims to minimize dependencies between modules and mitigate the impact of modifications to the system design. Late binding permits taking the right deployment decision as late as possible to generate communicating infrastructure at runtime. For addressing new requirements, functionality can be dynamically discovered, substituted, or integrated into the system in much the same way as swapping a reconfigurable module with another. The service orientation inherent dynamism is intrinsically compatible with the ability to dynamically reconfigure hardware modules exhibited by DRSoCs. Following these principles, we propose a SOCM-based framework for partial and dynamic reconfiguration modeling at transaction level. Indeed, a configuration will be dynamically composed of reusable modules providing services through dynamic bindings. Therefore, dynamic reconfiguration will be perceived as the result of the arrival or departure of services at runtime according to the publish-find-bind cycle. To develop Dynamically Reconfigurable Systems by minimizing

Table 2: Comparison with related works.

Ref	DPR support	UML/MARTE use	UML/MARTE DPR semantics covering		Design approach	Modeling paradigm	Target platform	Auto code gen	Transf. tools
			Extended packages	#new stereotypes					
[24]	No	UML-ESL	No	No	ESL	Service level	SystemC	Yes	N/A
[25]	No	UML/MARTE	No	No	MDD	Object-oriented	SystemC/TLM	Yes	UML2HIF
						Timing constraints	VHDL		HIF2SC/ HIF2VHDL
[26]	No	UML/MARTE	No	No	MDE	Aspect-oriented	VHDL	Yes	GenERTiCA
[30]	Yes	No	No	No	PRM	Process-based	SystemC	No	No
[31]	Yes	No	No	No	SW/HW codesign	Dynamic Module	SystemC/ TLM/RTL	No	No
[32]	Yes	No	No	No	System-level	Object-oriented	SystemC/TLM	No	No
[34, 35]	Yes	No	No	No	Y-chart based	Process-based CDFG	SystemC/TLM	No	No
[7]	Yes	UML/MARTE	No	No	MDA	Component-based	SystemC/ TLM/RTL	Yes	Acceleo/MTL
[38]	Yes	UML/MARTE	No	04	MDE	Component-based	RTL/VHDL/C	Yes	QVTO/Acceleo
[39]	Yes	UML/MARTE	No	05	MDE	Component-based	VHDL	Yes	N/A
[40–42]	Yes	RecoMARTE	HLAM, GCM, HRM, GRM	13	MDE	Component-based	Xilinx XPS	Yes	QVTO Acceleo
The proposed framework	Yes	MARTE4DPR MARTE4SCTLM	HLAM, GCM, SRM, HRM, GRM, GQAM	50	MDE	Service-oriented component	SystemC/TLM	Yes	ATL Acceleo

costs, the hardware resources should be allocated only when required, and services are thus lazily loaded. The SOCM-based reconfigurable component is composed of a factory unit and a controller, thus separating the business logic from cross-cutting concerns. Business logic or domain logic is the part of the code that implements the real-word business rules in terms of data manipulation. While the factory unit embeds the component functionality, the controller manages instance life cycle, service interactions, adaptation, bindings, persistence, migration, and security.

As depicted in Figure 1, after IP integration, the provider advertises service arrival by publishing its description at the service registry through which it is made dynamically discoverable. A service consumer can look up services by querying the service registry; it is also responsible for the selection and sorting of the available services to extract the best. The binding between service consumer and provider at TLM level can be performed by using initiator and target sockets through an interconnect component. A service consumer is notified about the arrival and removal of new and existing services, respectively.

By using the UML/MARTE and SystemC languages to raise the abstraction level of DPR modeling within a SOCM-based framework, we are inevitably faced with various challenges. The main challenge is how to model DPR in MARTE especially since the profile does not present any complete support. Another dilemma is that SOCM is not natively supported by MARTE besides the fact that it remains difficult to bridge the gap between UML/MARTE and SystemC/TLM domain-specific concepts. To address these challenges, we propose a framework layered architecture in which abstraction levels are built upon each other until reaching technical levels as shown in Figure 2. The SOCM principle implementation enabling support for dynamic availability of services is layered on the top of the architecture. MARTE is extended with DPR and SOCM concepts on the one hand and with SystemC/TLM concepts on the other hand giving rise to MARTE4DPR and MARTE4SCTLM profiles, respectively. TLM interoperability layer is composed of core interfaces, sockets, generic payload, and base protocol. The lower layer includes the SystemC library providing the core language, SystemC data types, predefined channels, and utilities.

4.1. High-Level Design Flow. The combined use of MDE model transformation techniques and MARTE foundations for model-based development helps to support a common design flow for real-time embedded systems with the integration of modeling and analysis tools. As depicted in Figure 3, we adopt the Y-chart-based codesign methodology to structure the design of DRSoCs and to handle their increased heterogeneity. The Y-chart flow is based on the separation of concerns principle; application and architecture models are built separately and subsequently associated in an allocation model. A set of front-end and back-end Eclipse-based tools are used to support MDE fundamental concepts. The dynamic reconfiguration issues are captured earlier by applying the MARTE4DPR profile at the top level.

Application modeling is based on composition of interacting component blocks enabling dynamic availability of services, whereas an architecture model is represented as a hierarchical structure of reconfigurable resources providing services to support the execution of the application. The functional elements are mapped onto architecture resources within an allocation model, which encompasses both spatial distribution and temporal scheduling features. From this allocation model which must be sufficiently detailed and precise, a SystemC/TLM simulation model can be obtained in two ways: either automatically through a model transformation following a rule-based approach or manually by the application of the MARTE4SCTLM profile. In turn, the resulting model is used for automatic SystemC/TLM code generation, which transforms a specification into an implementation according to a template-based approach. Finally, the execution of the generated code encompasses two major phases: elaboration followed by simulation. The simulation results can lead to a refinement of the specification models for performance improvement. To concretely perform the different model transformations, we have built a model-based toolchain from Eclipse IDE plugins [43]. Papyrus is used for creating and editing models, and it also provides support for UML profiles. We have chosen ATL and Acceleo tools to carry out M2M and M2T transformations, respectively. Source and target models are expressed in the XMI serialization format enabling the exchange of models between tools, while the metamodels conform to the ecore format, which represents a common ground ensuring reusability and interoperability.

4.2. Extending MARTE Profile. The intention of profiles is to give a straightforward mechanism for adapting an existing metamodel with constructs that are specific to a particular domain, platform, or method [8]. It is possible to extend profiles to create new adapted elements suitable for the modeling purpose.

To tailor the MARTE profile for covering DPR, SOCM, and SystemC/TLM domain concepts, we propose to extend it with specific constructs, stereotypes, and tagged values referenced in two profiles: *MARTE for DPR* (MARTE4DPR) and *MARTE for SystemC/TLM* (MARTE4SCTLM).

As illustrated by Figure 4, the MARTE4DPR package (stereotyped as profile) depends on HLAM, GCM, SRM, HRM, and GRM packages. It enables using adapted terminology, giving syntax and semantics to new types, and adding information used in model transformation. On the other hand, the MARTE4SCTLM package (stereotyped as profile) depends on HLAM, GCM, SRM, and GQAM packages; the intent is to provide specific stereotypes with iconic representation to map SystemC/TLM domain concepts. The proposed profiles enable the use of DPR domain-specific semantics and SystemC/TLM platform terminology while sharing all of MARTE's basic objectives. Thus, MARTE stereotypes can be reused by being referenced or specialized in MARTE4DPR and MARTE4SCTLM profiles. Users could apply the increment integrated or not with the

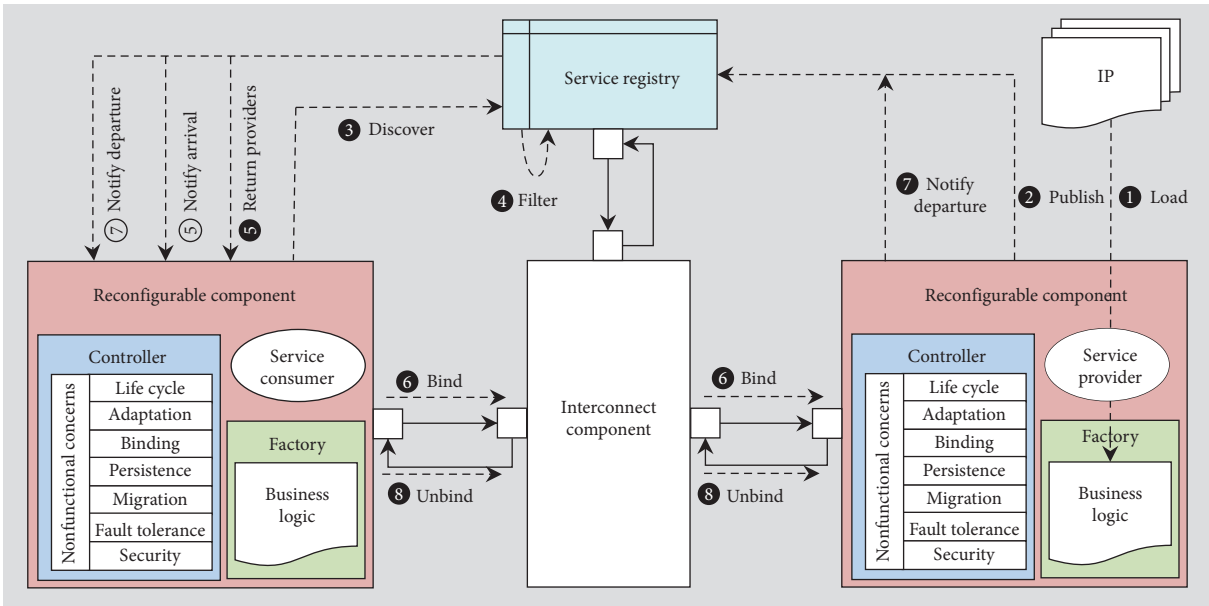

Figure 1: SOCM interaction pattern.

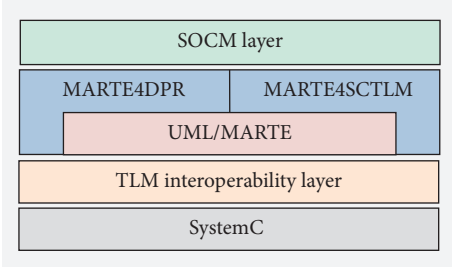

Figure 2: Framework layered architecture.

MARTE profile to address different concerns without conflicting constraints.

4.2.1. MARTE4DPR Profile. From the relevant starting points, MARTE stereotypes are refined to incorporate the missing DPR concepts. In what follows, we present the different extensions involving the MARTE subprofiles.

(1) Extending MARTE::HLAM. The concern of the HLAM package is to provide high-level modeling concepts to deal with real-time and embedded features modeling [17]. The related extensions are outlined in Figure 5.

The *ReconfigurableUnit* stereotype specializes the RtUnit (real-time unit) by adding several properties related to DPR and SOCM domains. An RtUnit is a high-level construct; it owns one or more schedulable resources which can be created dynamically and a specific state-based behavior in which states represent configurations and transitions characterize reconfigurations of the unit. A real-time unit can provide real-time services and invoke services of other real-time units. A *ReconfigurableUnit* maps a container element; it is composed of a *FactoryUnit* including the business logic and a *ReconfigControllerUnit* managing the component reconfiguration process and the port binding protocol. If a reconfigurable unit is dynamically and partially reconfigurable, its *isDynamicallyReconfigurable* and *isPartiallyReconfigurable* attributes must be true. The *mode* attribute determines the role of a component according to the TLM interoperability principle. Its literal values may be initiator, target, or interconnect. The *FactoryUnit* illustrates abstract factory pattern concepts describing a factory of factories. The abstract factory pattern provides an interface for creating families of related or dependent objects (products) without specifying their concrete classes allowing more decoupled and flexible design. Within a *FactoryUnit*, it is possible to substitute multiple factories to get multiple behaviors by creating specific products. The *ReconfigControllerUnit* manages the *FactoryUnit* instance life cycle, dependencies, transactions, and port binding. As depicted in Figure 6, a component instance enters the created state when it is launched for the first time without being ready for use. The configured composite state encompasses various service instance life cycle substates; as soon as the dependencies become available, the service will be resolved. After registration, the service component can be bound, and therefore the service is considered to be connected. Once the service function is spawned, the component enters the active state. When reconfiguration request or departure notification is triggered, the communication channels are locked and the component becomes passive. The stopped state is reached after a disconnecting request, and consequently the service instance would be unregistered.

The shared *ServiceRegistry* stereotype specializes the concept of protected passive unit (PpUnit). It provides permanent data storage for service metainformation including service description, IP configurations, and resource utilization parameters. The *IPCoreService* is an abstract concept that denotes the real-time behavioral features owned by a component. It could be specialized into

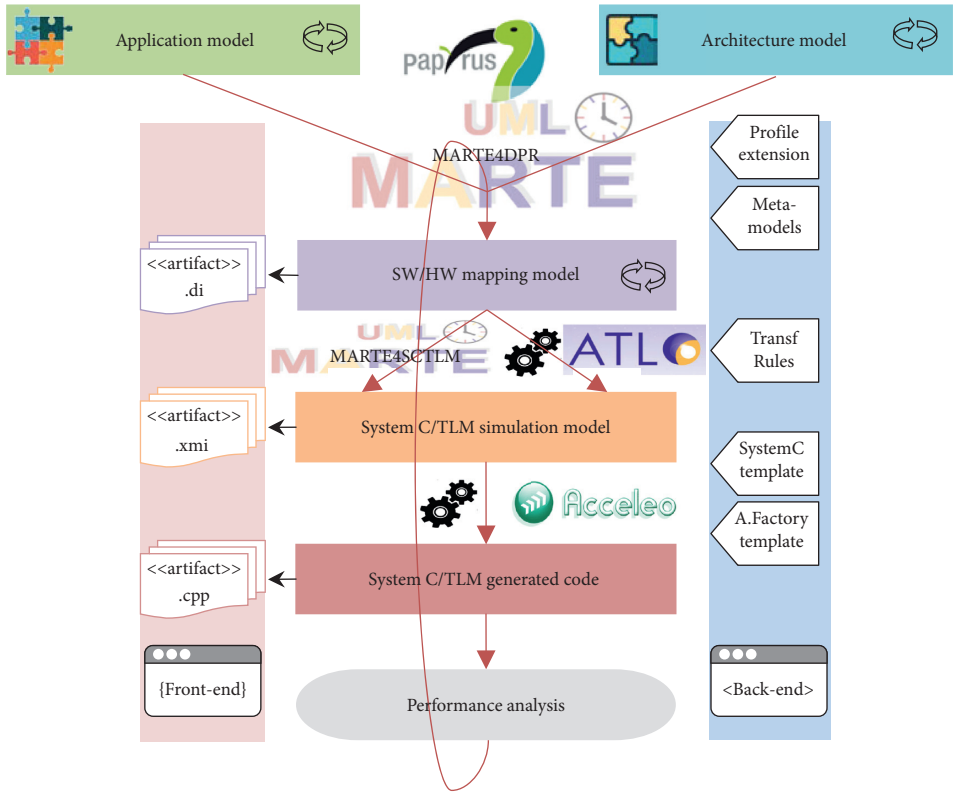

Figure 3: Y-chart design flow for DPR modeling.

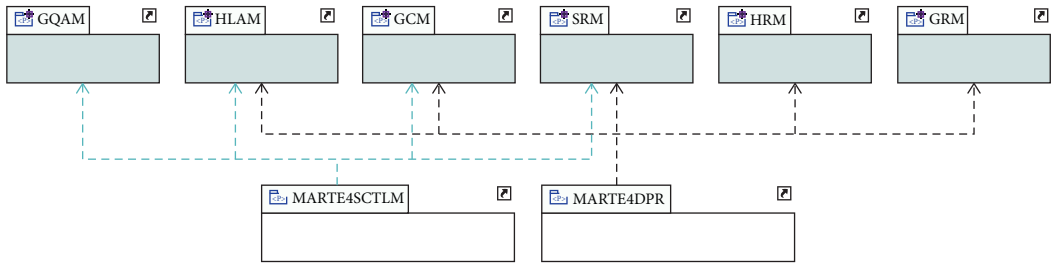

Figure 4: Dependencies of MARTE4DPR and MARTE4SCTLM packages.

SoftCoreService and *HardCoreService* stereotypes according to the type of core.

(2) Extending MARTE::GCM. The GCM package provides additional modeling concepts for real-time and embedded system component-based strategies. We mainly focus on particular refinements related to communication aspects to better catch dynamic reconfiguration domain concepts. Additional semantics on communication aspects require a specialization of the GCM semantics as depicted in Figure 7.

The main purpose of the *ReconfigurablePort/Export* stereotype is to support the interconnection evolution features of reconfigurable systems; it specializes the interface-based ClientServerPort stereotype. If its *isDynamic* attribute is set to true, then the interconnectivity may change at runtime. As shown in Figure 8, when a component reconfiguration (structural evolution) is triggered, all reconfigurable ports/exports have to be passivated and communication channels have to be blocked. Once the required connect/reconnect event is intercepted, the reconfigurable ports/exports will be reactivated. To manage correctly the component dependencies, the reconfiguration controller uses a local dependency table. A dependency is identified by the remote service component (provider) name, its port identifier, and the connection (binding) state.

(3) Extending MARTE::SRM. The SRM package is a specialization of resources and services defined in the GRM package; it focuses on the modeling of application programming interfaces of software multitasking platform. We are mainly interested here in modeling concurrent execution contexts described in the SW_Concurrency package.

As shown in Figure 9, the *FactoryProcess* stereotype defines a business processing context owned by a factory unit; it specializes the SwSchedulableResource stereotype in order to support the modeling of static/dynamic spawned

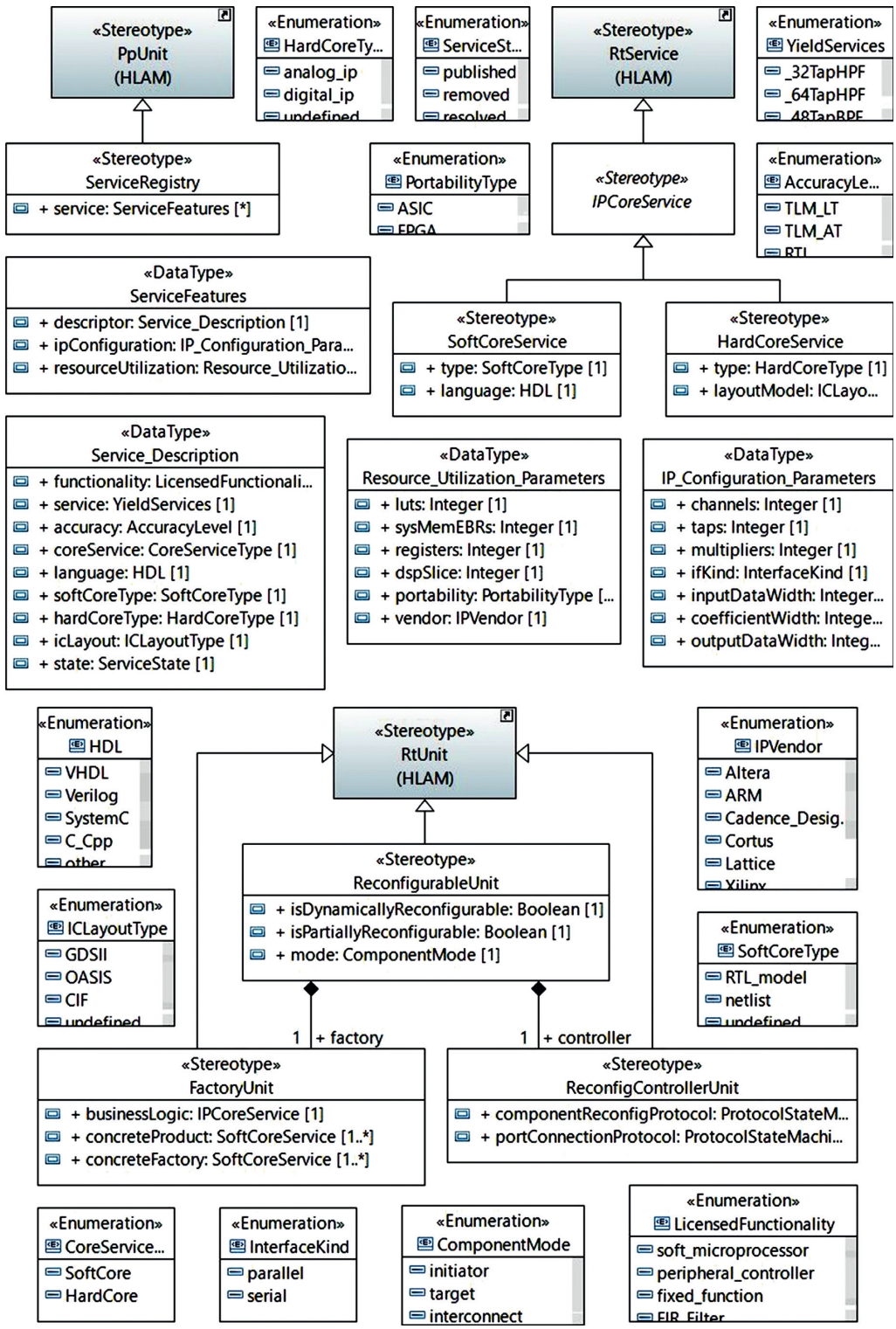

Figure 5: Extended HLAM subprofile.

processes. The *spawnedFunction* attribute represents the function associated with the spawned process instance. The *ReconfigurationControllerProcess* stereotype provides specific services to dynamically manage and supervise the reconfiguration process; it specializes the SwSchedulableResource stereotype to support services dependencies resolution, reconfiguration triggering, and service publication and discovery.

(4) Extending MARTE::GRM. The GRM package provides the concepts that are necessary to model a general platform for executing real-time embedded applications. For the

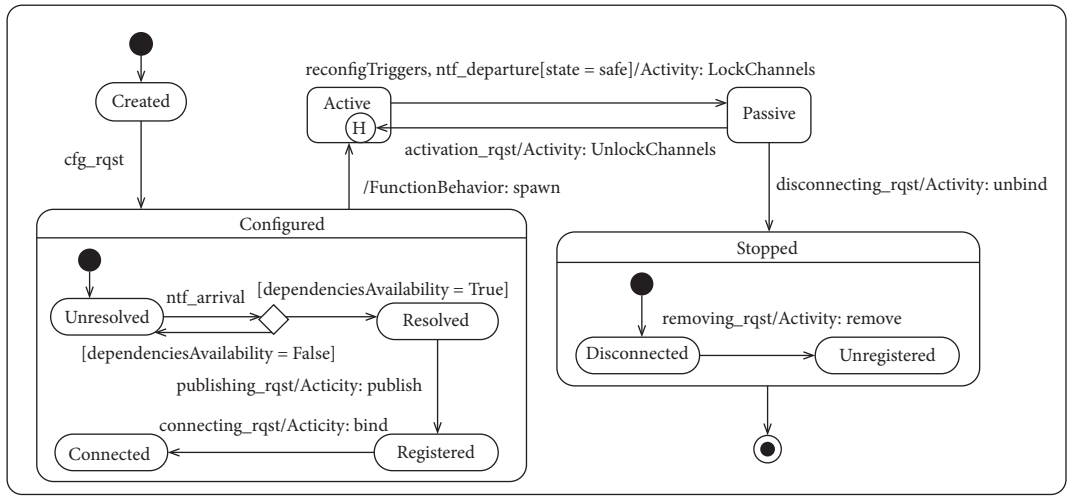

Figure 6: Component instance life cycle.

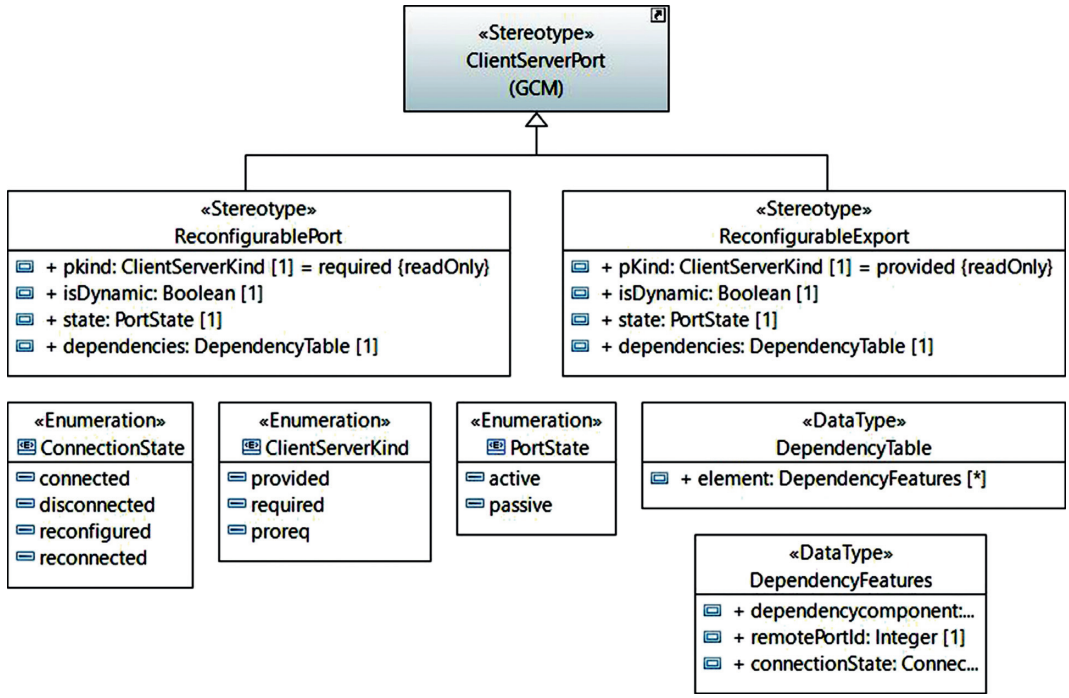

Figure 7: Extended GCM subprofile.

purpose of software and hardware modeling, the GRM is specialized to SRM and HRM, respectively. To support reconfiguration features, the GRM may be further specialized.

A *Resource* represents a physically or logically persistent entity that offers one or more resource services. Resources are used to model the execution platform from a structural point of view, while the resource services supply the behavioral point of view [17]. Since the notion of Resource is a central concept of the GRM, it is redefined within MARTE4DPR by adding a Boolean attribute *isReconfigurable* which expands the semantics of resource and propagates it through its types as depicted in Figure 10.

(5) Extending MARTE::HRM. The HRM package is grouping most hardware concepts under a hierarchical taxonomy with several categories depending on their nature, functionality, technology, and form [17]. Separation of concerns and abstraction are the main qualities of this profile. The HRM is composed of two complementary and converging views: a logical view that provides a functional classification of hardware entities based on services that each resource offers and a physical view that concentrates on their physical properties such as shape, size, and position within platform, power consumption, and heat dissipation. To support partial and dynamic reconfiguration features, it would be interesting to extend semantics of stereotypes from logical model

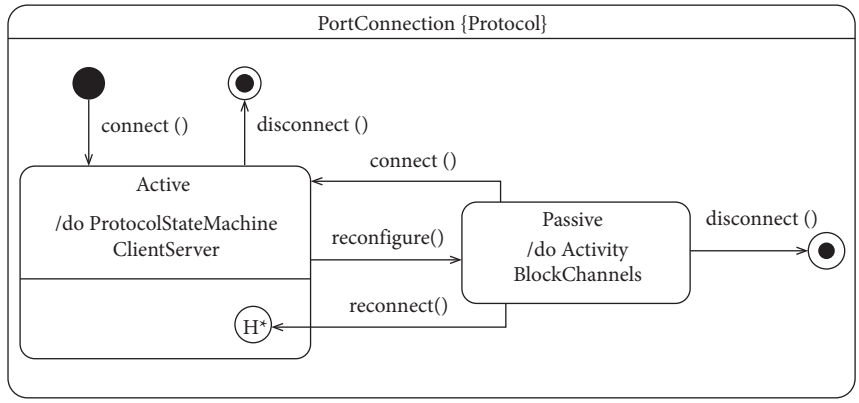

Figure 8: ReconfigurablePort/Export connection protocol.

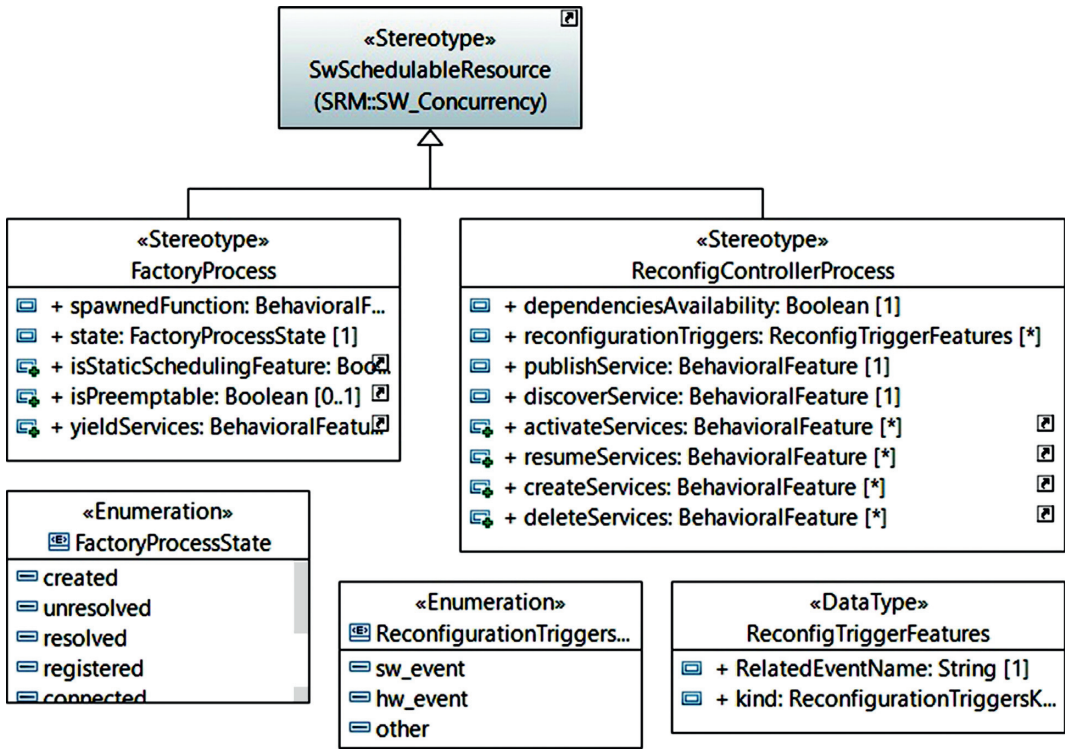

Figure 9: Extended SRM subprofile.

having their physical correspondents such as HwLogical::HwComputing::HwPLD and HwPhysical::HwLayout::HwComponent as described in Figure 10.

The HW_Layout package provides hardware component classification depending on their forms and offers arrangement constructs using rectilinear grids [17]. The HwComponent stereotype maps the HW_Component domain element which is the main physical entity of the HW_Layout package. It is specialized by the *ReconfigHwComponent* stereotype owning additional attributes related to reconfiguration features. If an HwComponent is dynamically and partially reconfigurable, the attributes *isDynamicallyReconfig* and *isPartiallyReconfig* must be set to true. Both *isActive* and *isReconfigurable* attributes are inherited from Resource and declared read only. *isActive* means that it has its own course of action which allows it to perform its services autonomously.

The HwPLD is a programmable computing resource; it has a special organization and it may own several IPs, hardwired or not, such as processors, memories, and analogic devices [17]. The *FPGA* stereotype is a specialized HwPLD which contains other specific attributes. The *granularity* attribute takes its values in the enumeration *GranularityType*; the dynamic relocation capability is indicated by the Boolean attribute *isRuntimeReloc*. The FPGA *regions* attribute denotes the features of the regions in terms of dimensions and kind (reconfigurable or static). While the *designFlow* attribute specifies the dynamic and partial reconfiguration design flow style, the *configInterface* attribute indicates the FPGA configuration access interface modes.

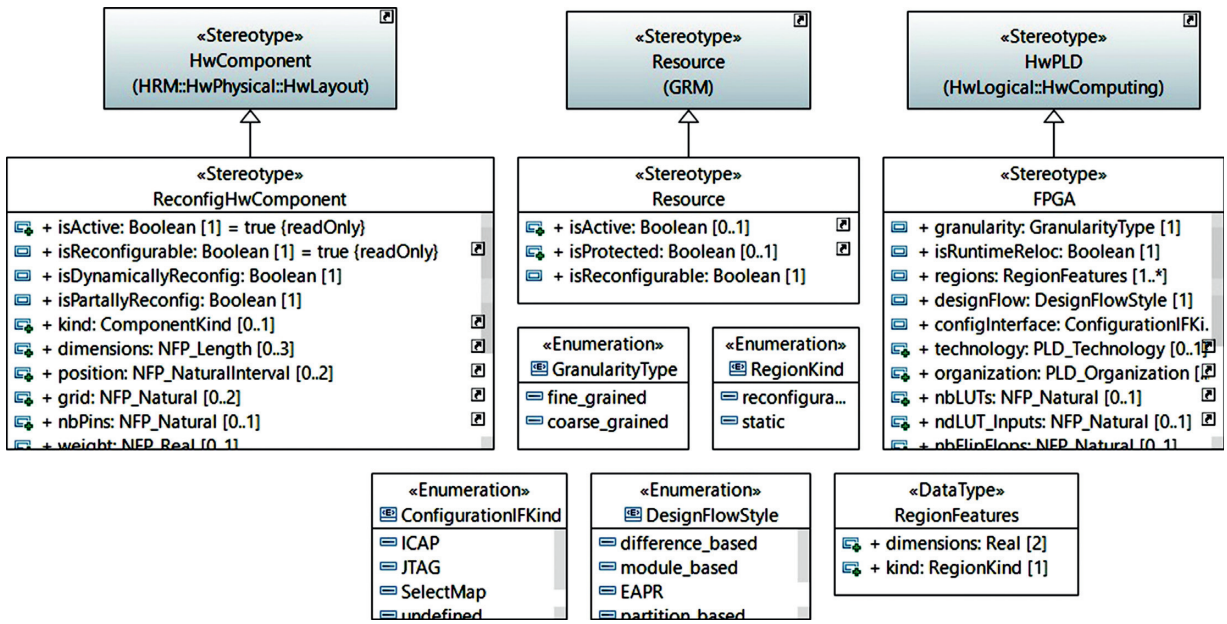

Figure 10: Extended HRM GRM subprofiles.

4.2.2. MARTE4SCTLM Profile. A SystemC module is the smallest container of functionality with state, behavior and structure for hierarchical connectivity [44]. The *Sc_Module* stereotype maps the SC_Module domain element. It specializes the HLAM::RtUnit stereotype with additional attributes; the component role within TLM interoperability layer is specified by the *role* attribute that takes its values in the enumeration {initiator, target, interconnect, top}. The *variables* and *helpers* attributes define local member variables and helper functions declared in the *Sc_Module*. The image associated with that stereotype is . The whole profile is presented in Figure 11.

SC_PrimChannel is a SystemC construct used for implementing processes communication and modules interconnection. The *Sc_Prim_Channel* stereotype specializes the MessageComResource stereotype of SRM::SW_Interaction package with specific semantics appropriate to the transaction-level modeling. Indeed, the MessageComResource defines communication resource to exchange messages (structure of data) which is compliant with TLM payload event queue semantics. The *sensitivityList* and *triggerList* attributes enable specifying passed arguments of the member functions wait and next_trigger, respectively.

In SystemC, a process instance can be created by invoking sc_method, sc_thread, or sc_cthread macros or by calling the sc_spawn function. It is possible to recognize spawned, unspawned, static, and dynamic processes according to the execution phase callback and function/macro from which the process instance is created. Dynamic spawned processes are the most appropriate to model and simulate dynamic reconfigurable systems. Indeed, in such systems, elements are generated or eliminated while the system is running which can be naturally specified by dynamic spawned processes that are created from the end_of_elaboration callback or during simulation. The SwSchedulableResource stereotype from SW_Concurrency subprofile is specialized by the *Sc_Process* and *Sc_Spawn* stereotypes. The *Sc_Process* concept is abstract; it is extended by *Sc_Method*, *Sc_Thread*, and *Sc_Cthread* stereotypes; when it is dynamically spawned, its attributes *isDynamic* and *isSpawned* must be set to true. The static sensitivity of the process instance is specified by the *sensitivityList* attribute, while dynamic sensitivity is denoted within the extended stereotypes by *waitArgs* and *nextTriggerArgs* attributes. The *function* attribute defines the member function associated with the process instance. The *Sc_Spawn* stereotype is applied to the functions to be spawned as processes. If its *isDynamic* attribute is set to true, the process instance is dynamically spawned. The process or module from which the sc_spawn function is called is specified by the parent attribute, whereas the *options* attribute denotes possible process instance properties. The image is associated with the *Sc_Spawn* stereotype.

TLM involves communication between processes using interface method calls through ports and exports. A port (respectively, export) defines a set of services that are required (respectively, provided) by the module containing the port (respectively, export). Sockets are used to pass transactions between initiators and targets. Technically, an initiator socket is derived from class sc_port and has a sc_export, and vice versa for a target socket. On the other hand, the MARTE ClientServerPorts support a request/reply communication paradigm and specify a set of provided/required services, as well as the type of produced/consumed signals represented by messages. For these reasons, in MARTE4SCTLM, the GCM::ClientServerPort stereotype is specialized to *Sc_Port* and *Sc_Export* stereotypes which in turn are, respectively, extended by *TLM_Initiator_Socket* and *TLM_Target_Socket* stereotypes. These latter are also specialized by *Simple_Initiator_Socket* and *Simple_Target_Socket* stereotypes to map particular sockets which enable dynamic processes spawning. The *module* attribute

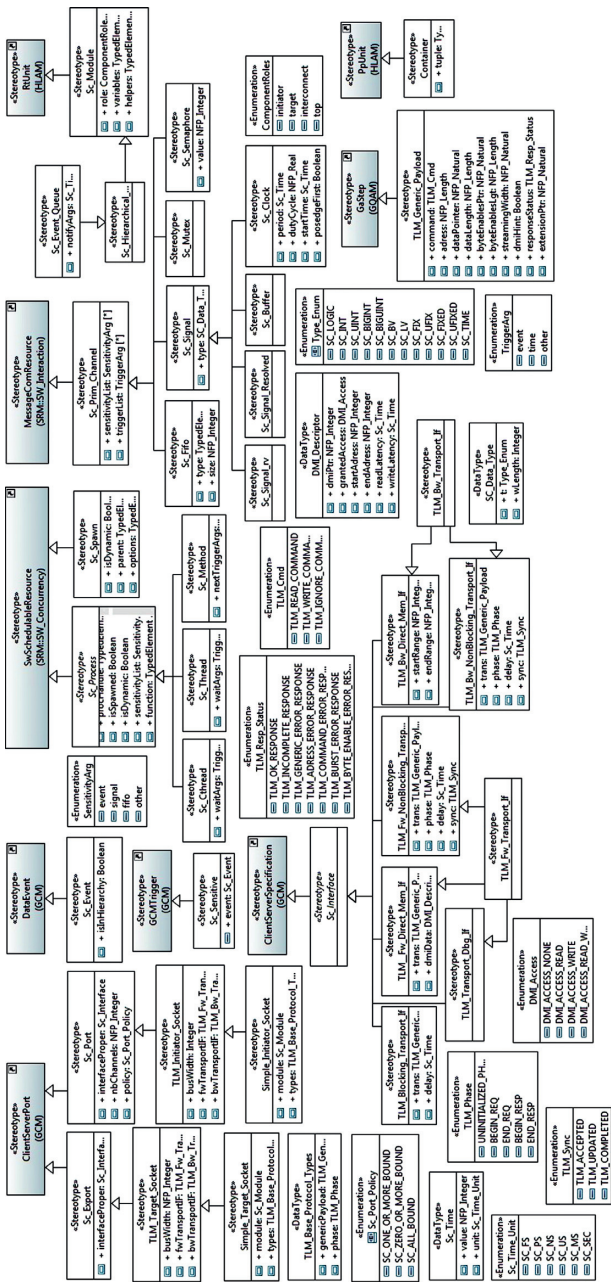

Figure 11: MARTE4SCTLM profile.

specifies the socket owner component, whereas the *types* attribute allows initiator/target socket to be parameterized with the default protocol type including the generic payload and base protocol phases.

The TLM-2.0 core interfaces consist of the blocking and nonblocking transport interfaces, the direct memory interface (DMI), and the debug transport interface. While the blocking transport interface only uses the forward path from initiator to target and supports the loosely timed coding style, the nonblocking transport interface uses both the forward path from initiator to target and the backward path from target to initiator and supports the approximately timed coding style. The Sc_Interface concept matches the MARTE concept of ClientServerSpecification. The *Sc_Interface* abstract stereotype is specialized to multiple stereotypes corresponding to the core interfaces. The *TLM_Fw_NonBlocking_Transport_If* and *TLM_Bw_NonBlocking_Transport_If* stereotypes specialize two distinct nonblocking interfaces for use on opposite paths; they have four additional attributes to specify the transaction type, phase type, timing annotation, and synchronization return value.

The generic payload can be used as a general-purpose transaction type for abstract memory-mapped bus modeling or as the basis for modeling a wide range of specific protocols at a more detailed level. To maximize interoperability between models, the *TLM_Generic_Payload* default transaction type should be used with the base protocol. Within MARTE4SCTLM profile, the *TLM_Generic_Payload* stereotype specializes the GQAM::GaStep stereotype with a standard set of attributes (see TLM2 LRM [45] for more details).

The *Sc_Event* stereotype specializes the GCM::DataEvent stereotype; if the event is a hierarchically named event, the Boolean attribute *isInHierarchy* must be set to true. Each process instance can have its own static sensitivity established immediately after its registration. The *Sc_Sensitivity* stereotype maps the semantics of static sensitivity by specializing GCM::GCMTrigger stereotype; the *event* attribute denotes the set of events that would cause a process to be resumed or triggered.

The *Container* stereotype maps C++ container; it specializes the HLAM::PpUnit stereotype for model transformation.

5. Evaluation and Results

This section presents an experimental validation of our approach through a case study. We discuss how the proposed framework has been applied for a stepwise refinement of specification models at different abstraction levels. Metrics are then used for assessing the quality of model transformations.

5.1. Case Study. We chose to illustrate our proposal with a very practical case study relating to a reconfigurable active 3-way crossover. As shown in Figure 12, a crossover is a network that filters input audio sound signals by frequency, diverting the low-frequency sounds to a woofer, mid-frequency sounds to a midrange speaker, and high-frequency sounds to a tweeter. Incorporating the three speakers makes it possible to have the best sound from the full audio spectrum.

A 3-way crossover can be performed by means of three types of pass filters: a high-pass filter (HPF) allows higher frequencies to pass through the filter, a band-pass filter (BPF) passes all frequencies between upper and lower frequencies, and a low-pass filter (LPF) allows lower frequencies to pass through the filter. The finite impulse response (FIR) filter is used to implement high-pass, band-pass, and low-pass filters. It is characterized by the number of taps, the transition frequencies, and the sampling

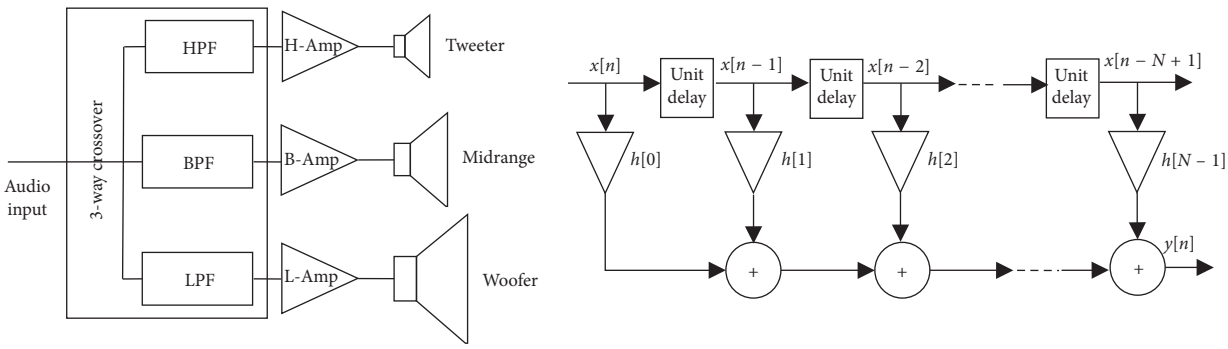

Figure 12: Active 3-way crossover and N-tap FIR filter logical structure.

frequency of the audio signal to be filtered. The input-output signal relationship of an N-tap FIR filter is described by the following equation:

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k], \tag{1}$$

where $h[k]$ are the filter coefficients, $x[n-k]$ represent the input samples, and $y[n]$ is the output signal. Each input sample is multiplied by a coefficient, and then the resulting products are added together to give one output sample. According to the convolution sum formula (1), Figure 12 shows typical FIR filter logical structure. The set of possible IP configurations with resource utilization information is given in Table 3.

Regarding the reconfigurability of the FIR filter, the dynamism and substitutability of services can be perceived at two different levels:

(1) Behavioral reconfiguration: Since the different filter types have the same structure with different behaviors, it is possible to implement them by one reconfigurable component (service provider) able to substitute functionality dynamically when it is required.

(2) Architectural reconfiguration: To meet performance and energy consumption requirements, it is possible to update the number of filter taps. An N-tap filter can be substituted by an M-tap filter of the same filter type. This is reflected in the hardware by a partial reconfiguration of the original filter.

5.1.1. Application Model. For reasons of flexibility, the MARTE4DPR and the MARTE4SCTLM profiles can be applied to all structural UML diagrams. However, we used the composite structure diagram to depict the internal structure of classifiers and their interactions with the environment, which is suitable for both hardware modeling and SCA developing. As shown in Figure 13, from a functional point of view, the simulated active 3-way crossover is made up of several parts. Two instances of dynamic and partial reconfigurable FIR filter called Filter#1 and Filter#2 are stereotyped as *ReconfigurableUnit*. Filter#1 is composed of Controller#1 and Factory#1 components stereotyped as *ReconfigControllerUnit* and *FactoryUnit*, respectively. Factory#1 provides a FIR filter service and its concrete implementation, namely, 32TapHPF and 64TapHPF products. To handle the execution of its spawned 64TapHPFservice, Factory#1 includes FactProc#1, a dynamically schedulable resource stereotyped by *FactoryProcess*. Controller#1 is responsible for managing the reconfiguration process according to specific protocols related to the FIR filter reconfiguration modes and the reconfigurable port connection states. It encompasses a dynamically schedulable resource stereotyped as *ReconfigControllerProcess*, enabling the initiation of a reconfiguration process once the rcfgRqst event has been triggered and service dependencies have been satisfied. Furthermore, CtrlPro#1 provides specific services to manage publication, discovery, activation, resumption, and passivation of services. Processes embedded within Filter#1 communicate through stereotyped reconfigurable ports and exports using interface method calls. Services can be published and discovered by means of a centralized protected passive unit called Catalog and stereotyped *ServiceRegistry*. The DummyAudioDriver and DACStub real-time units emulate behaviors of audio driver and digital-to-analog converter unit, respectively. To avoid cluttering the model, it is worth mentioning that Filter#2 is structurally identical to Filter#1 but behaves differently by providing other services. The basic transport service is provided by the interconnect component which represents a logical bus abstraction and is stereotyped as *CommunicationMedia*. Components can then make use of the transport service via communication ports. When model elements are refined by applying stereotypes, it becomes necessary to assign values to their properties. Figure 14 shows Papyrus dialog boxes which enable setting the service feature values published in the Catalog.

The modal behavior of a reconfigurable component can be modeled by a state diagram where a mode represents a particular configuration. A mode transition can be produced in response to a trigger related to an event verifying the reconfiguration conditions. Figure 15 shows that a filter has three possible modes: HighPass, LowPass, and BandPass for which a reconfiguration is triggered when the corresponding condition is verified. A filter can be implemented as N-tap or M-tap depending on the required performance level. It can be dynamically switched from one mode to another.

Table 3: FIR filter configurations set.

Config	Taps	Multipliers	Channels	Coef_Width	LUTs	SysMem_EBRs	Registers	Dsp_Slice
#1	24	6	1	32	241	4	272	1
#2	32	8	1	32	201	4	303	9
#3	32	32	1	32	202	2	199	6
#4	48	12	1	32	246	4	281	1
#5	64	1	4	32	242	3	306	6

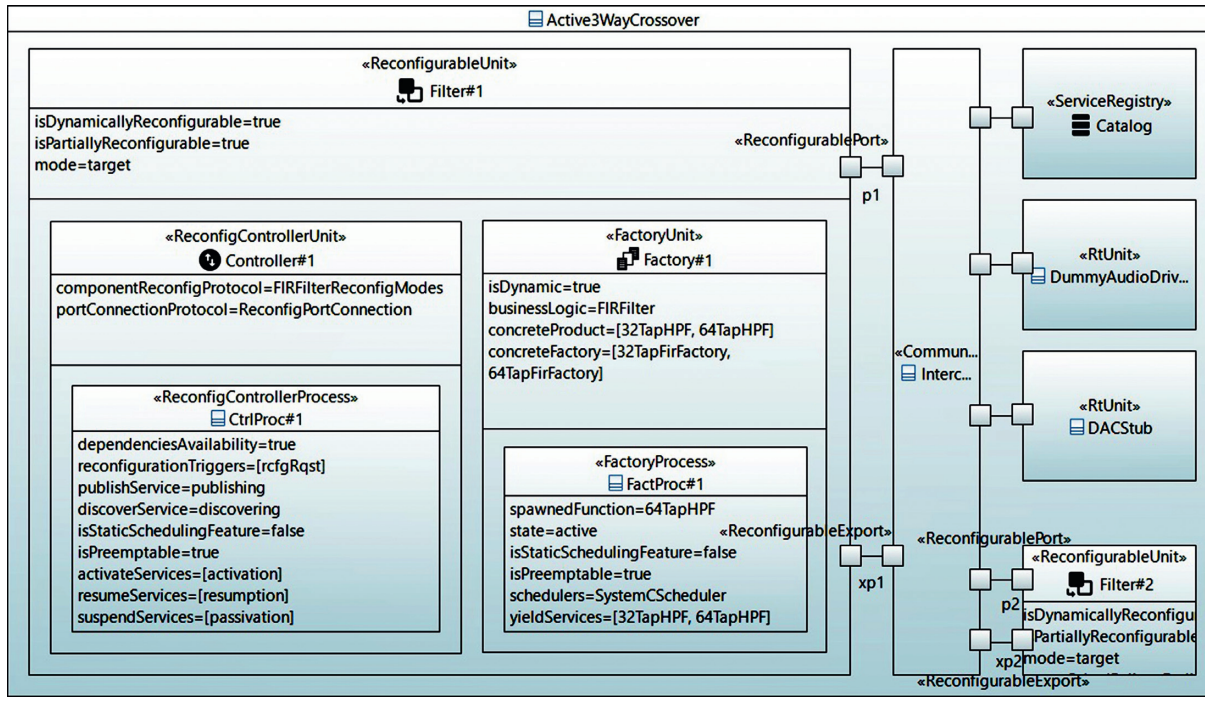

Figure 13: High-level abstract view.

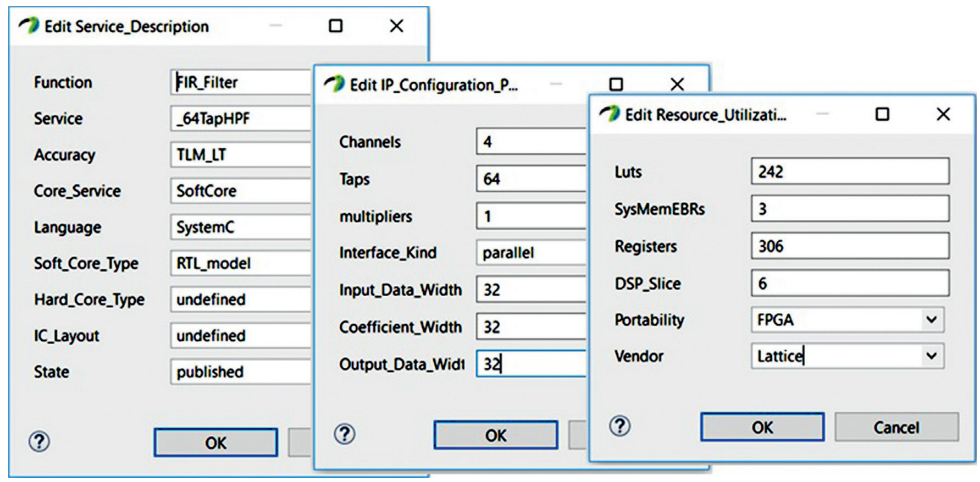

Figure 14: Assigning values to service attributes.

5.1.2. Execution Platform and Mapping Model. Within MARTE, hardware platform can be modeled through two complementary views: a logical view that distinguishes hardware resources according to their functional role and their provided/required services; a physical view that is interested in their physical properties including shape, size, position, power consumption, and heat dissipation. Since logical and physical views converge on many concepts, they could be merged into a unified view.

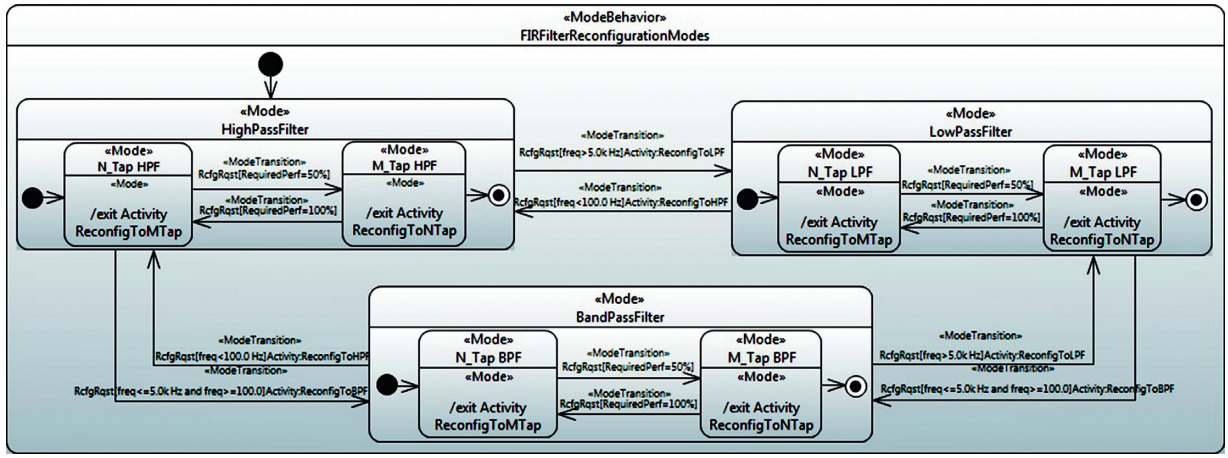

Figure 15: Filter structural/behavioral reconfiguration.

As depicted in Figure 16, the Active3WayCrossover component is refined into more specialized one by applying the HwComponent and the *FPGA* stereotypes. The component is a card with physical properties such as the Cartesian dimensions, the associated rectilinear grid, and the environmental requirements. The FPGA stereotype specifies a fine-grained reconfigurable architecture with runtime relocation ability. The DPR is carried out using partition-based design flow through ICAP interface. The spatial distribution of reconfigurable regions is defined by the regions multivalued attribute. The component is annotated by the number of LUTs, inputs of a LUT, and flip-flops. The owned computing blocks and the range of supported frequencies are also specified. Filter#1 is stereotyped as *ReconfigHwComponent*; it is a dynamically and partially reconfigurable chip characterized by layout and power properties including its position, static consumption, and static dissipation. Factory#1 subcomponent is stereotyped with HwComputingResource and *ReconfigHwComponent* denoting an active execution resource dynamically reconfigurable. However, the HwComputingResource and HwComponent stereotypes are applied to Controller#1 to capture the resource processing capabilities within a container. The Catalog component is stereotyped as HwMemory and HwComponent to describe a data storage resource offering read/write services. The interconnect component is stereotyped as HwBus and HwComponent; it defines particular hardware media (channel) characterized by its address and word widths, its temporal properties, and transmission mode.

5.1.3. SystemC/TLM Simulation Model. A SystemC/TLM simulation model is obtained by applying the MARTE4SCTLM profile to the allocation model as depicted in Figure 17. The top-level component Active3WayCrossover is stereotyped as *Sc_Module*; its *isMain* attribute is set to true meaning that the module will be instantiated within sc_main function.

To construct the module hierarchy, the *Sc_Module* stereotype is applied to Filter#1 component to create a submodule within the parent module. Its *role* attribute is set to target specifying that the module will act as a target in TLM modeling. The variables and helpers attributes are part of data members and member functions to be declared within the *Sc_Module*. Factory#1 and Controller#1 nested components are also stereotyped as *Sc_Module*. The former contains a special member function called 64TapHPF stereotyped as *Sc_Spawned* and *Sc_Thread* to handle the creation of a spawned process instance called from a thread process, and the latter holds a member function stereotyped as Sc_Thread to map the control service. The 64TapHPF function is dynamically spawned from FactProc#1 thread and may be substituted by the 32TapHPF function according to the sensitivity list. Ports and exports are, respectively, stereotyped as *Simple_Initiator_Socket* and *Simple_Target_Socket* to support both forward and backward paths corresponding to a sequence of method calls. Once the *TLM_Fw_Transport_If* stereotype is applied, its attributes must be explicitly initialized, which enables capturing relevant information for automatic code generation later, as described in Figure 18. The DACStub is stereotyped as Sc_Module and plays the role of initiator. Finally, the Catalog is stereotyped as *Container* to specify a data structure providing management and access functions.

5.1.4. Automatic Code Generation. The concept of transformation is fundamental to MDE; it consists of a refinement process decreasing the abstraction level of models by adding enough details to mapped models for automatic code generation. Depending on the nature of the target model, it is possible to distinguish between model-to-model (M2M) transformations and model-to-text (M2T) transformations. To implement our toolchain, we combined a M2M transformation to produce a SystemC/TLM target model from a MARTE4DPR source model with a M2T transformation to translate the previously produced model into a SystemC/TLM source code.

In our framework, M2M transformation is expressed by means of the Atlas Transformation Language (ATL): a domain-specific language and toolkit developed on the top of Eclipse platform. The input model precisely specified in

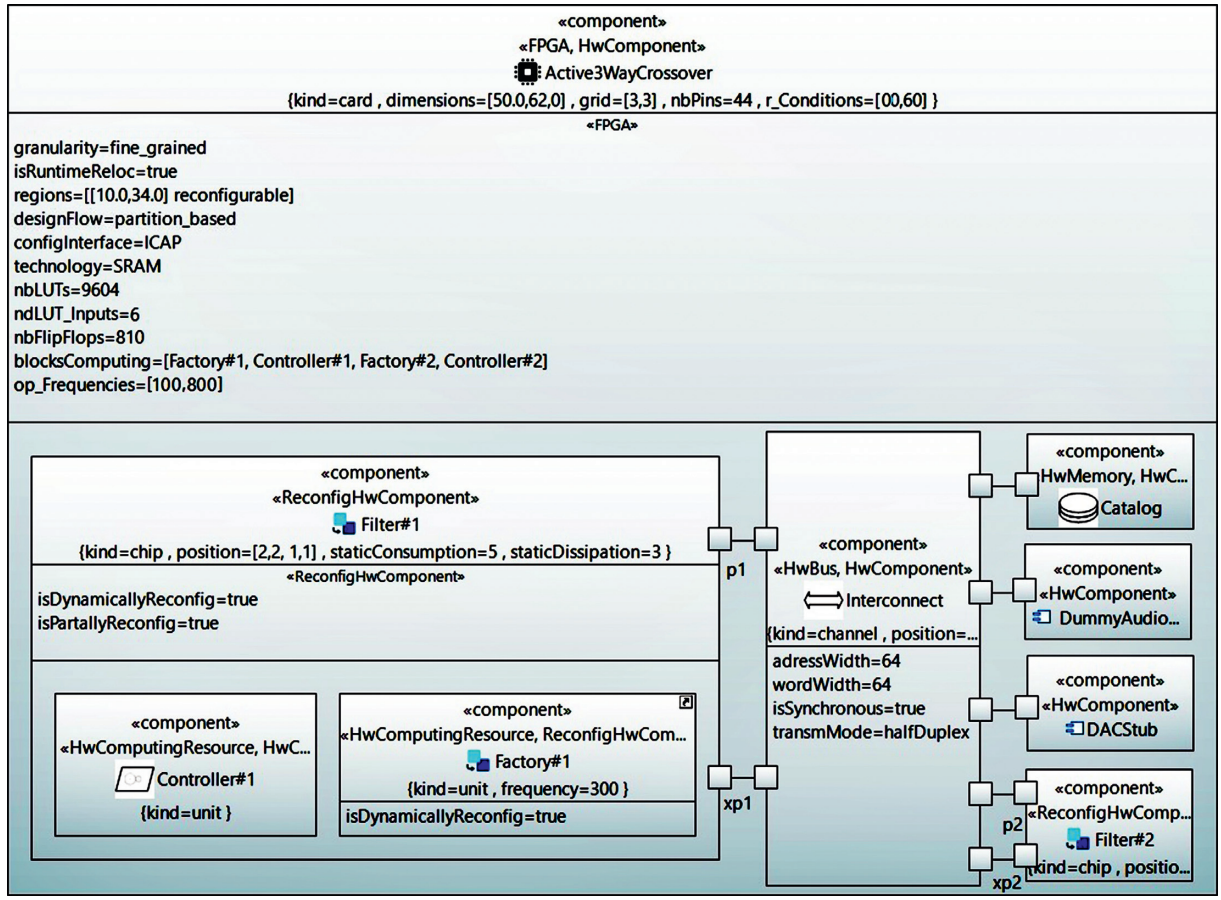

Figure 16: Merged logical physical view.

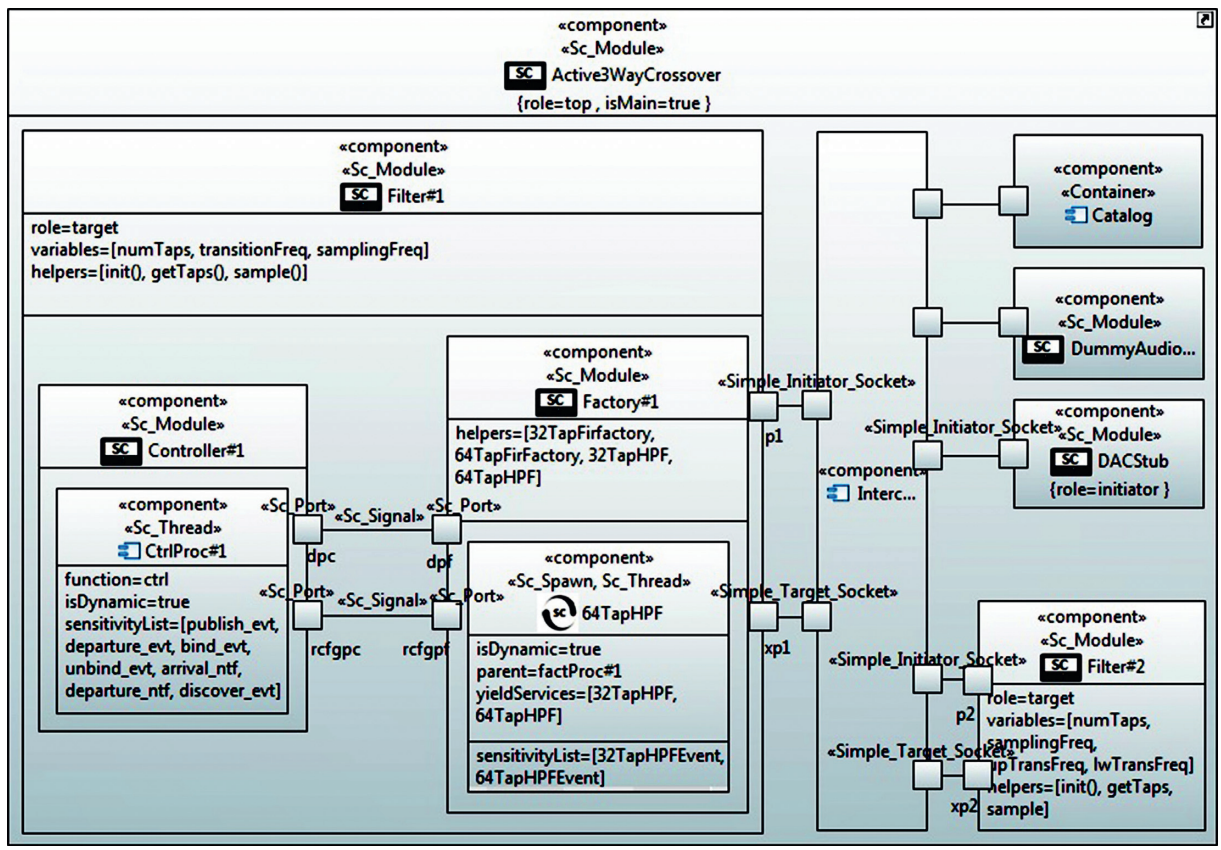

Figure 17: SystemC/TLM simulation model.

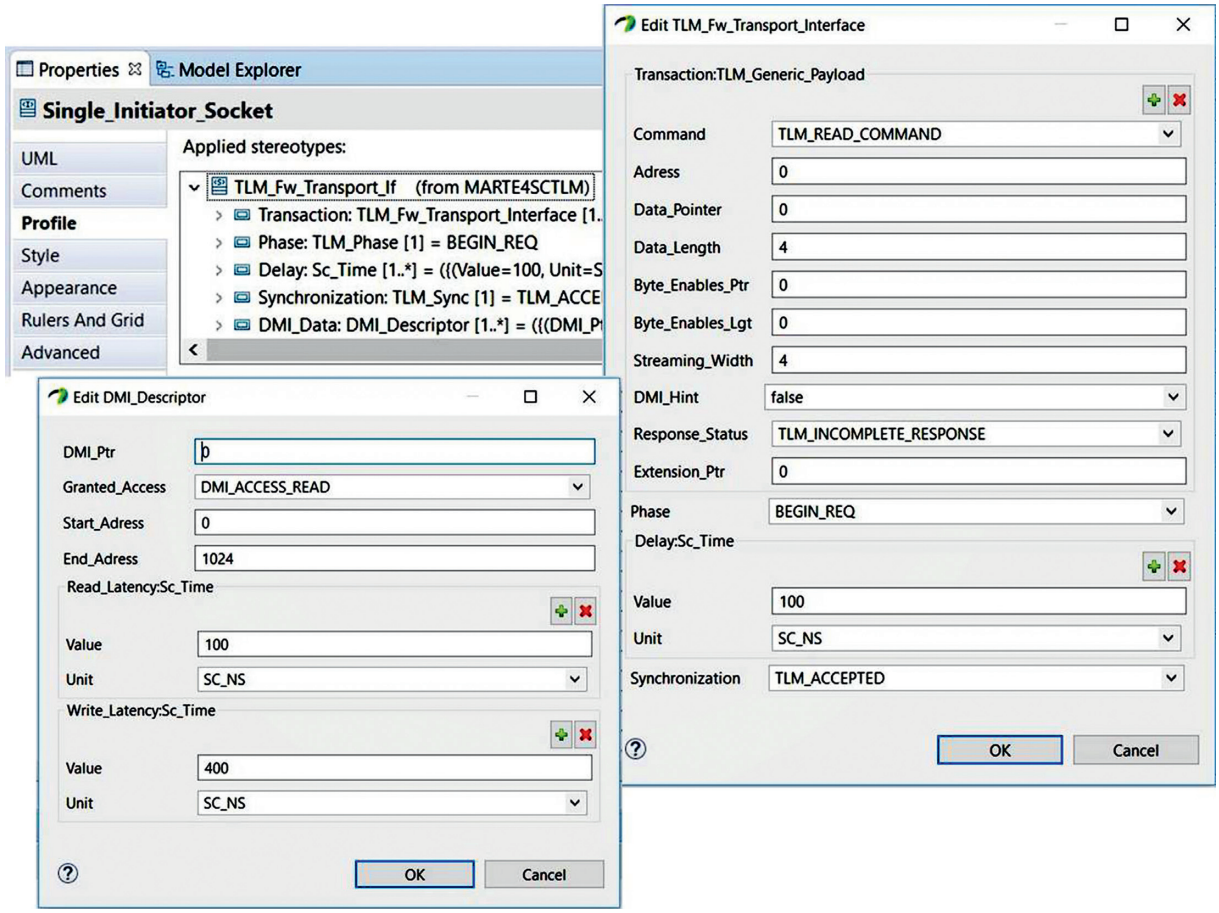

Figure 18: Assigning values to the TLM_Fw_Transport_If stereotype attributes.

the MARTE4DPR metamodel is transformed into a target model that conforms to SystemC/TLM metamodel by following a common model transformation pattern. Eclipse Modeling Framework (EMF) provides support to create and instantiate ecore metamodels. In Figure 19, we used the EMF tree-based editor to show a part of the hierarchy of the MARTE4DPR metamodel with a conforming model instance.

Since ATL is a textual rule-based language, the generation of SystemC/TLM model elements is performed by applying declarative rules written at the metamodel level. In order to specify these matched rules, it is necessary to find similarities between source and target model concepts to relate the metamodels. Despite semantics variations between UML and SystemC, it is still possible to reduce the gap by building a mapping relationship of similar elements between UML/MARTE, MARTE4DPR, and SystemC/TLM.

The mapped concepts have been chosen so that they are semantically close to one another. In the context of DPR, the convergence between UML/MARTE and SystemC/TLM can be achieved through MARTE4DPR profile because of its conformance with UML/MARTE on the one hand and to the common domain of interest with SystemC, i.e., the real-time and embedded domain, on the other hand. Table 4 shows the mapping from UML, MARTE, and MARTE4DPR concepts into SystemC/TLM ones, taking into account DPR structural and behavioral semantics.

Once the declarative rules are triggered, the MARTE4DPR model elements should be matched and navigated in order to create and initialize the elements of the target model. As depicted in Figure 20, the ATL transformation module is composed of helpers, model elements, and standard matched rules (written in declarative style). Both isSpawned and isDynamic helpers are defined in the context of the FactoryProcess element and used to calculate Boolean values stating whether a factory process is dynamically spawned. The ReconfigurableUnit2SCModule rule is intended to generate SystemC module from a reconfigurable unit. The attributes of the generated Sc_Module are initialized using bindings. Other transformation rules as well as the generated SystemC/TLM model are presented in Appendix A.1.

Now that the semantics is precise enough, the application code can be generated by transforming models into textual artifacts. A template-based M2T transformation is specified and built with the Acceleo language. This transformation specification is structured into two modules that generate SystemC/TLM source code and C++ abstract factory pattern implementation.

The first generation engine takes as input the previously produced SystemC/TLM model which is in compliance with the metamodel shown in Figure 21. An Acceleo module

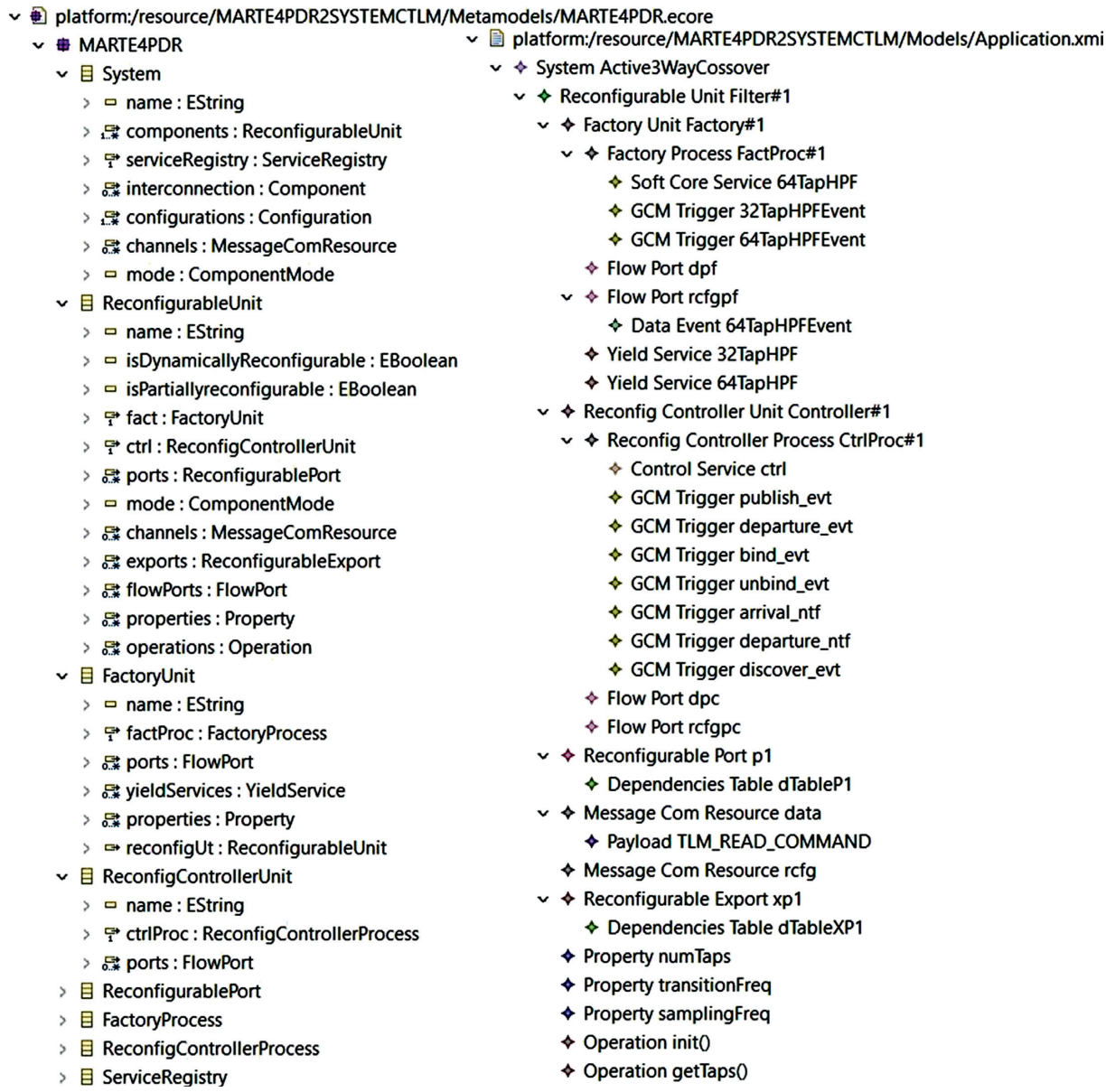

Figure 19: The MARTE4DPR metamodel (left) and the application model (right).

contains templates made up of static and dynamic parts specifying text files with placeholders to be filled with data extracted from the input model. Figure 22 depicts a part of a related template used for generating the object hierarchy within a SystemC module.

The second engine is dedicated to the code generation of business logic according to the abstract factory design pattern. The generated code contains template classes and services declaration to be paired with code files (refer to Appendix A.2).

5.2. Experimental Results. Benefiting from the advantages of Eclipse-based tools, our framework offers facilities for graphical expressiveness with highly customization, extensibility, reusability, and maintainability capabilities. Through more than 50 new stereotypes to cover missing DPR concepts and SystemC/TLM constructs in MARTE, the proposed extensions should ensure the profile completeness with almost no modeling overhead. Furthermore, Papyrus tool provides extensive support facility for defining and applying increments conforming to the UML/MARTE standard. Existing tools and knowledge can be reused, thus reducing learnability effort and tooling cost. In order to demonstrate the effectiveness of the proposed framework with regard to produced artifacts, a comparative analysis between the automatic code generation and hand-coded implementation of the system is proposed. The performance was measured on a Intel(R) Core (TM) i5-8265U running at 1.6 GHz with 4 GB RAM under Windows 10. Table 5 summarizes the quantitative evaluation of the hand-coded implementation of the system calculated with LocMetrics tool [46]. The system implementation has a total of 1256 lines of code spread over 6 source files requiring 6.041

Table 4: Concept mapping.

UML concepts	MARTE concepts	MARTE4DPR concepts	SystemC/TLM concepts
Active class	HLAM::RtUnit {isDynamic=true}	ReconfigurableUnit	SC_Module
Active class	HLAM::RtUnit {isDynamic=true}	FactoryUnit	SC_Module
Active class	HLAM::RtUnit {isDynamic=true}	ReconfigControllerUnit	SC_Module
Operation, StateMachine	SRM::. . .::SwSchedulableResource	FactoryProcess	SC_Thread, SC_Method
Operation, StateMachine	SRM::. . .::SwSchedulableResource	ReconfigControllerProcess	SC_Thread, SC_Method
Port	GCM::ClientServerPort	ReconfigurablePort	TLM_Initiator_Socket TLM_Target_Socket
Interface	GCM::ClientServerSpecification	ClientServerSpecification	SC_Interface
Class	HLAM::PpUnit	ServiceRegistry	C++ Container
BehavioralFeature	HLAM::RtService	IPCoreService	C++ Function
DataType	VSL::DataTypes::TupleType	ServiceDescriptor	C++ Tuple
Message	GQAM::. . .::GaStep	Payload	TLM_Generic_Payload
Class, connector	SRM::. . .::MessageComResource	MessageComResource	SC_Prim_Channel
Event	GCM::DataEvent	DataEvent	SC_Event
Trigger	GCM::GCMTrigger	GCMTrigger	SC_Sensitive
Class	HRM::HwLogical::. . .::HwPLD	FPGA	SC_Module
Class	GRM::Resource {isActive=true}	Resource	SC_Module
Class	HRM::. . .:: HwComponent	ReconfigHwComponent	SC_Module
BehavioralFeature	GRM::GrService	GrService	C++ function

```
1  -- @path MM=/MARTE4PDR2SYSTEMCTLM/Metamodels/MARTE4PDR.ecore
2  -- @path MM1=/MARTE4PDR2SYSTEMCTLM/Metamodels/SYSTEMC_TLM.ecore
3  module Rules;
4  create OUT : MM1 from IN : MM;
5  -- The following helpers can retrieve the values of the boolean attributes
6  -- of a factory process.
7  -- They return true if the owner module is dynamically reconfigurable.
8  helper context MM!FactoryProcess def: isSpawned(): Boolean =
9         self.factoryUt.reconfigUt.isDynamicallyReconfigurable = true;
10 helper context MM!FactoryProcess def: isDynamic(): Boolean =
11        self.factoryUt.reconfigUt.isDynamicallyReconfigurable = true;
12
13 -- The following rule generates a SystemC module from a reconfigurable unit.
14 rule ReconfigurableUnit2SCModule {
15        from
16               reconfigUt : MM!ReconfigurableUnit
17        (reconfigUt.oclIsTypeOf(MM!ReconfigurableUnit))
18        to
19            scMod : MM1!SC_Module (
20                       name <- reconfigUt.name,
21                       role <- reconfigUt.mode,
22                       sc_module_factory <- reconfigUt.fact,
23                       sc_module_controller <- reconfigUt.ctrl,
24                       initiatorSocket <- reconfigUt.ports,
25                       targetSocket <- reconfigUt.exports,
26                       primeChannel <- reconfigUt.channels,
27                       ports <- reconfigUt.flowPorts,
28                       variables <- reconfigUt.properties,
29                       helpers <- reconfigUt.operations  )
30 }
```

Figure 20: Fragment of the ATL transformation module.

person-months of development effort. With a measured value of 117 for the cyclomatic complexity, the code appears to be very difficult to maintain and test. In addition, hand coding is well known to be a very tedious, error prone, and time-consuming process.

For assessing the internal quality of ATL model transformation, we used a set of specific metrics based on quality attributes introduced in [47, 48]. These latter include inter alia understandability, modifiability, reusability, modularity, completeness, consistency, and complexity. We observe in Table 6 that only 286 lines of code are required to perform the transformation, making it more understandable, more modifiable, and less complex. There are 26 matched rules each linked to an input element and 2 helpers with a low complexity code. The input/output metamodels coverage rate is high (96.29% and 100%, respectively), which is a good indicator of the model transformation completeness. The transformation is specified in a declarative style hiding details of encoding relations between source and target patterns. This should increase the

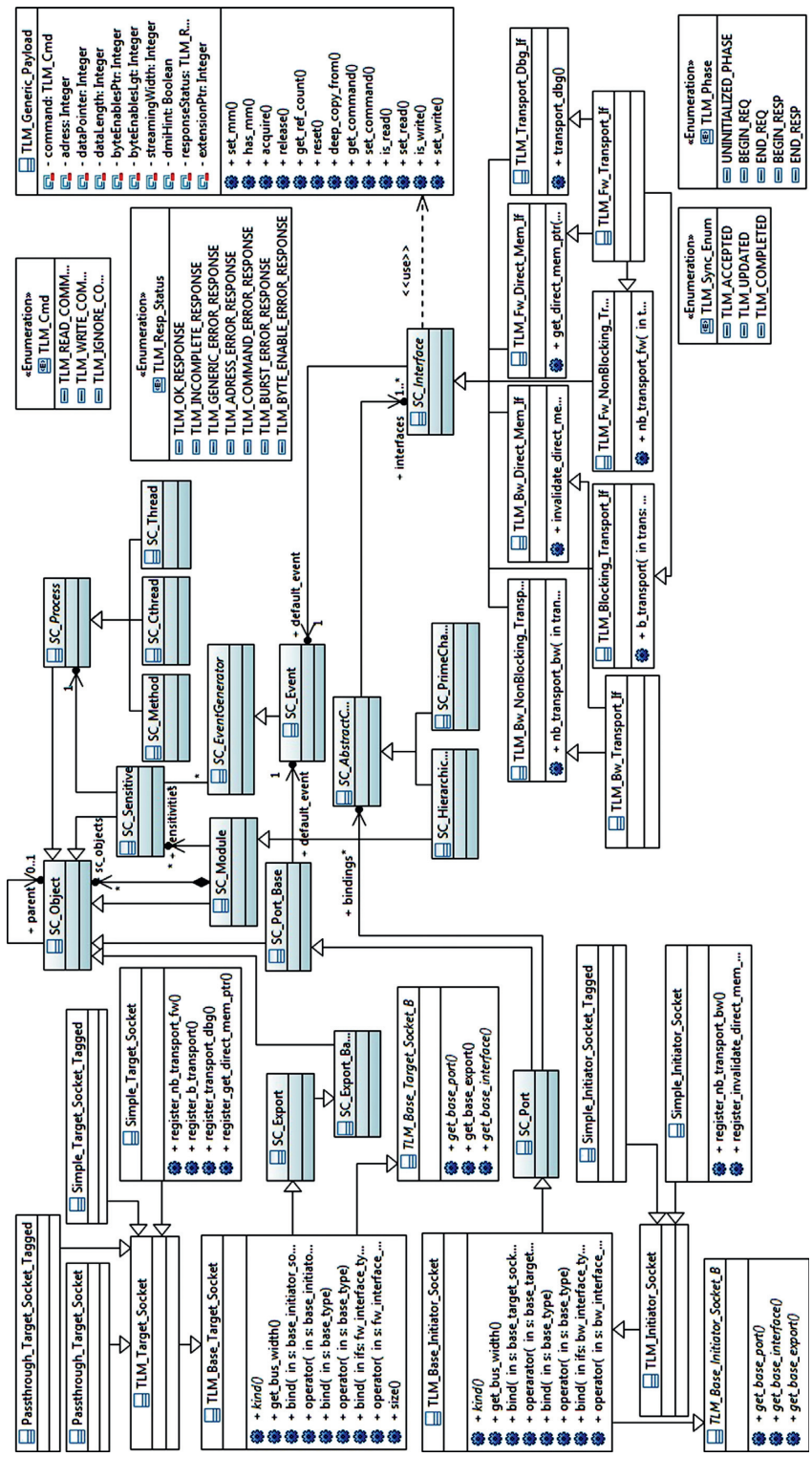

Figure 21: SystemC/TLM metamodel.

understandability and modifiability of transformation algorithms.

As depicted in Figure 23, the results suggest that our approach requires fewer LOC and therefore less development effort than hand coding. Moreover the increase of total LOC according to the number of components in the system is much lower in our approach than in hand coding. Similar results are provided when analyzing the relationship between LOC and the number of reconfigurations (see Figure 24).

The Acceleo built-in profiler gives the ability to keep track of evaluations and identify bottlenecks in generation process. As shown by the profiling data in Figure 25, the time

```
SC_MODULE([it.sc_module_factory.name/])
  {
  [for (itpf :SC_Port | it.sc_module_factory.ports)]
  sc_in<[itpf.type/]> [itpf.name/];
  [/for]
  [for (ite : SC_Event | it.sc_module_factory.ports.event)]
  sc_event [ite.name/];
  [/for]
  void [it.sc_module_factory.threads.function.name/]()
  {
  sc_spawn(sc_bind(&[it.sc_module_factory.threads.spawn.function.path/]::[it.sc_mod
  ule_factory.threads.spawn.function.name/],this));
  };
  SC_CTOR([it.sc_module_factory.name/])
  {
  SC_THREAD([it.sc_module_factory.threads.function.name/]);
  sensitive [for (its : SC_Sensitive |
  it.sc_module_factory.threads.sensitivity)]<<[its.name/][/for];
  }
  };
SC_MODULE([it.sc_module_controller.name/])
  {
  [for (itpc : SC_Port | it.sc_module_controller.ports)]
  sc_in<[itpc.type/]> [itpc.name/];
  [/for]
  [for (itec : SC_Event | it.sc_module_controller.ports.event)]
  sc_event [itec.name/];
  [/for]
  };
  [for (itf : Function | it.sc_module_controller.threads.function)]
  void [itf.name/]([for (itpar : Parameter | itf.parameters)][itpar.type/]
  [itpar.name/][/for])
  {
  [if (itf.name='publishService')]
  [anEClass.containers.name/].push_back([itf.parameters.name/]);
  [else]
  [if (itf.name='deleteService')]
  [anEClass.containers.name/]['['/]srvName[']'/].state='Removed';
  [/if]
  [/if]
  }
  [/for]
  SC_CTOR([it.sc_module_controller.name/])
  {
  [for (itth :SC_Thread | it.sc_module_controller.threads)]
  SC_THREAD([itth.function.name/]);
  sensitive [for (itsen : SC_Sensitive | itth.sensitivity)]<< [itsen.name/][/for];
  } };
```

Figure 22: Part of the template for SystemC/TLM code generation.

Table 5: The measurement results of the hand-coded implementation.

Metric	Value
Source files	6
Directories	2
LOC, lines of code	1256
BLOC, blank lines	91
SLOC-P, physical executable lines of code	1024
SLOC-L, logical executable lines of code	759
McCabe VG complexity	117
C&SLOC, code and comment lines of code	79
CLOC, comment only lines of code	141
CWORD, commentary words	1272
HCLOC, header comment lines of code	12
HCWORD, header commentary words	27

spent on generating source code for two reconfigurable components is 336 ms, while less than 90 s is required to customize and adjust it to meet the final application requirements.

Aligning MDE practices with the fundamental concepts of SW/HW codesign leads to a shorter and less expensive development life cycle, compared to traditional approaches (see Figure 26). The semantic enrichment of models by additional annotations and constructs to perform transformations results in an elongated application/execution platform modeling phase. On the other hand, by starting the automation process, the entire life cycle will be impacted and the following phases will be logically shortened. Thus, the accurate models obtained previously help to save development time and reduce the effort of the implementation

Table 6: Evaluation of the ATL transformations.

Metric	Value	Quality attributes						
		A1	A2	A3	A4	A5	A6	A7
# lines of code	286	+	+					−
# lines of comments	8	+						
Balance of a unit	1	+		+	+			
# matched rules	26	+	+					−
# lazy matched rules	0							
# called rules	0							
Average number of bindings per rule	3.42	+	+			−	+	
# rules with a filter condition on an input pattern	1	+	+					
Rule complexity increase	0.69	+						−
# helpers	2	+	+	−				
# calls to OCL functions	1	+	+					
Average helper cyclomatic complexity	1	+						
# input models	1		+	+				−
# output models	1		+	+				
Input metamodel coverage	96.29					+		+
Output metamodel coverage	100					+		+
Transformation approach	Declarative	+	+					−

A1: understandability; A2: modifiability; A3: reusability; A4: modularity; A5: completeness; A6: consistency; A7: complexity; +: positive effect; −: negative effect.

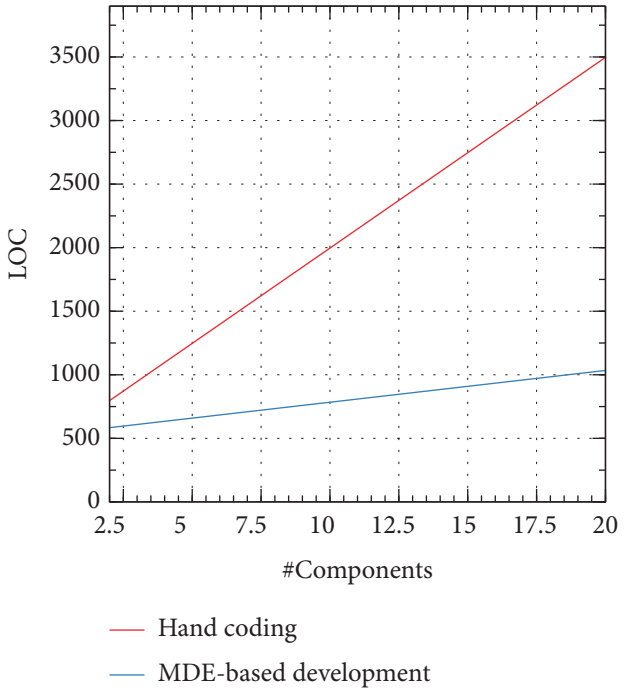

Figure 23: Relationship between LOC and the number of components.

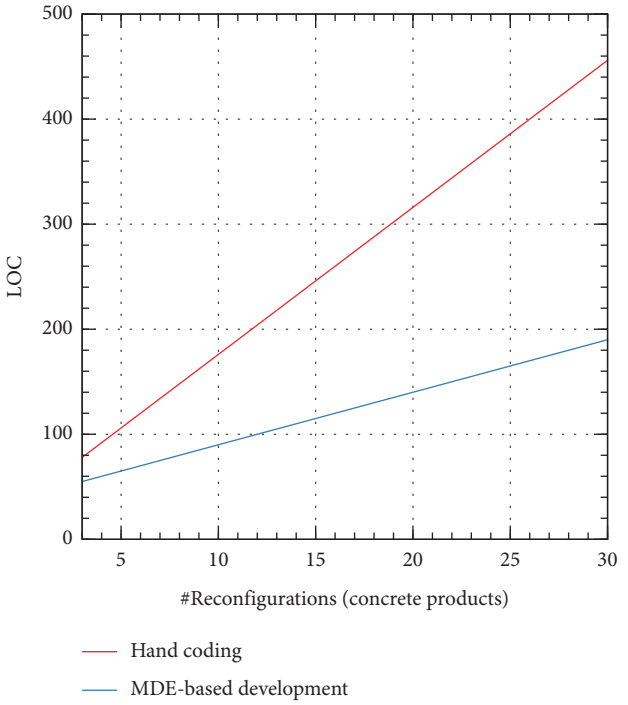

Figure 24: Relationship between LOC and the number of reconfigurations.

compared to manual coding. Transparent design-runtime traceability information provides a sound support for artifacts tracking and verification. Indeed, a SystemC class can be traced back to its UML/MARTE stereotyped class allowing easier verification and test automation with faster feedback on performance analysis. A code customization phase may be required to adjust and update the generated code according to different design requirements. As we expected, despite requiring a little more modeling effort, our

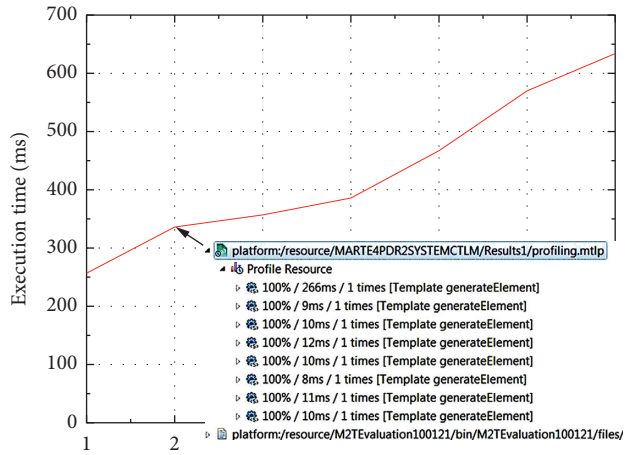

Figure 25: The generation process profiling results.

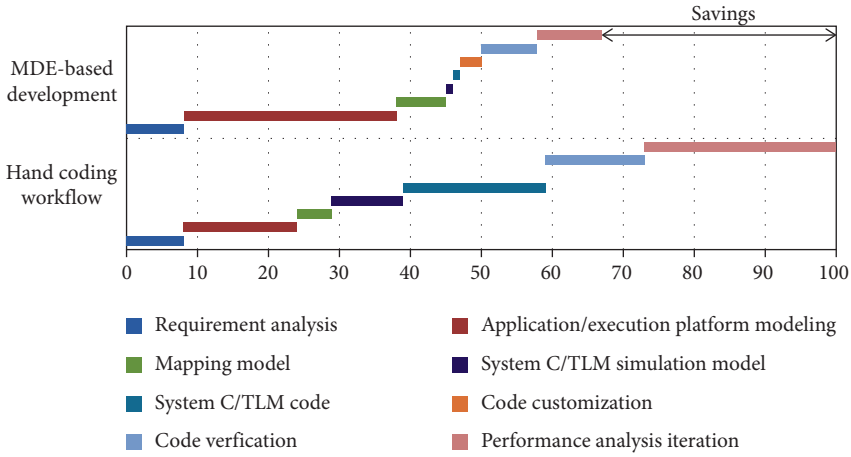

Figure 26: Classical hand coding workflow compared to MDE-based development process for DRSoCs design.

approach results in average time savings of 33% with significant design productivity gain.

6. Conclusion

In this paper, we have presented a high-level modeling framework for Dynamically Reconfigurable Systems-on-Chip targeting SystemC at transaction level. To better leverage software engineering solutions in terms of dynamic availability and service adaptation support, a service-oriented component-based approach was applied to DRSoC modeling. For this purpose, we have proposed an MDE-based design flow using extended MARTE models as initial specification. The latter are progressively refined to reach a SystemC/TLM executable code.

One of the main contributions of this work is the extension of MARTE profile with missing modeling capabilities. Indeed, although dedicated to model-driven development of embedded systems, MARTE needs to be specialized with both DPR and service orientation semantics, which has been realized in the MARTE4DPR profile. Several packages modeling both hardware platforms and software applications are involved in the extension. Corresponding MARTE stereotypes are then sufficiently refined for precise and complete DPR and SCA domain-specific concept modeling. In addition, MARTE stereotypes have been further specialized to bridge the gap between higher-level specifications and lower-level SystemC/TLM implementation giving rise to the MARTE4SCTLM profile while ensuring consistency from design to runtime.

Concretely, to provide automation for modeling reconfigurable SoCs and speed up simulation model generation, a completely integrated Eclipse toolchain was built to ensure support for creating and editing models, applying profiles, and executing transformations for code generation. Moreover, to perform the required model transformations, a set of rules expressed in ATL and specified in declarative style was provided. The final output is produced instantly through an Acceleo template-based code generation.

In order to assess the proposed framework, our choice fell on the specification of a reconfigurable active 3-way crossover. Particular emphasis was given to the modeling of FIR filter reconfiguration features through different views and detail levels as well as code generation. Profile extensions supply the standard with a complete set of stereotypes and tagged values, undeniably bringing more expressiveness

```
-- Hierarchical module creation.
-- The following rule generates the top level module.
rule System2SCModule {
        from
                sys : MM!System
        to
                scMod : MM1!SC_Module (
                        name <- sys.name,
                        role <- sys.mode,
                        subModules <- sys.components,
                        subModules <- sys.interconnection,
                        containers <- sys.serviceRegistry,
                        primeChannel <- sys.channels  )
}
-- The following rule generates a SystemC module from a factory unit.
rule FactoryUnit2SCModule {
        from
                factUt : MM!FactoryUnit
        to
                scMod : MM1!SC_Module (
                        name <- factUt.name,
                        process <- factUt.factProc,
                        ports <- factUt.ports,
                        configuration <- factUt.yieldServices,
                        variables <- factUt.properties  )
}
-- The following rule generates a Sc_Thread from a factory process.
rule FactoyProcess2SCThread {
        from
                factProc : MM!FactoryProcess
        to
                scThread : MM1!SC_Thread (
                        name <- factProc.name,
                        isSpawned <- factProc.isSpawned(),
                        isDynamic <- factProc.isDynamic(),
                        function <- factProc.services,
                        sensitivity <- factProc.associatedEvents  )
}
```

Figure 27: Some ATL transformation rules.

and completeness to the models. Successive model transformations maintaining traceability between artifacts result in decreasing significantly design complexity and development effort.

In future work, we aim to improve our framework by IP-XACT modeling capabilities for component description which is very useful for dynamic service discovery and service reuse modeling. The conflict between predictability and dynamism in service component architecture will be considered to ensure system correctness. Further, we intend to integrate our framework into vendor tools to implement DPR features on specific FPGAs and benefit from interoperability, scalability, substitutability, low maintenance, and rapid prototyping of SCA in DRSoCs, thus extending our transformation chain to reach an accurate RTL model for reconfigurable component implementation through MDE principles. Some challenges related to DPR can be addressed at high abstraction level by enriching the framework back end with the necessary profile extensions, metamodels, and transformation rules for intelligent DPR control, resource partitioning, task scheduling, and context saving/restoring. The integration of our framework with vendor tools should allow us to better address low-level device dependent operations such as decoupling logic, placement and routing, and bitstream relocation. Naturally, having more detailed models at lower levels of abstraction will come at the cost of faster simulation speed and earlier availability.

We also plan to explore the potential of an emerging approach called big service to address the challenges of adaptive services in the context of Dynamically Reconfigurable Systems. Indeed, big service is a complex and evolving service ecosystem that deals with the Big Data [49]; it allows dynamically creating composite services based on customer requirements specification across various domains, networks, and cyber-physical worlds.

Appendix

A. Code Generation Process

A.1. ATL Transformation Rules and Model Generation. Figures 27 and 28, respectively, show some transformation rules and a part of the generated SystemC/TLM model in XMI format whose semantics is sufficiently specified to have a better code quality later.

```
<subModules name="Filter#1">
  <helpers name="init()"/>
  <helpers name="getTaps()"/>
  <helpers name="sample()"/>
  <variables name="numTaps" type="Int" value="64"/>
  <variables name="transitionFreq" type="Float" value="3.0"/>
  <variables name="samplingFreq" type="Float" value="44.1"/>
  <initiatorSocket name="p1">
    <Dependencies dependencyComponent="DummyAudioDriver" remotePort="pDUM"
    connectionState="Connected"/>
  </initiatorSocket>
  <targetSocket name="xp1">
    <Dependencies dependencyComponent="DACStub" remotePort="pDAC"
    connectionState="Connected"/>
  </targetSocket>
  <primeChannel name="data">
    <payload id="2" command="TLM_READ_COMMAND" adress="32" dataPointer="32"
    dataLength="4" streamingWidth="4" responseStatus="TLM_OK_RESPONSE" dmiHint="false"/>
  </primeChannel>
  <primeChannel name="rcfg"/>
  <sc_module_factory name="Factory#1">
    <process xsi:type="SYSTEMC_TLM:SC_Thread" name="FactProc#1" isSpawned="true"
    isDynamic="true">
      <function name="64TapHPF"/>
      <sensitivity name="32TapHPFEvent"/>
      <sensitivity name="64TapHPFEvent"/>
    </process>
    <ports name="dpf"/>
    <ports name="rcfgpf">
      <event name="64TapHPFEvent"/>
    </ports>
    <configuration yieldService="32TapHPF"/>
    <configuration yieldService="64TapHPF"/>
  </sc_module_factory>
  <sc_module_controller name="Controller#1">
    <process xsi:type="SYSTEMC_TLM:SC_Thread" name="CtrlProc#1">
      <function name="ctrl"/>
      <sensitivity name="publish_evt"/>
      <sensitivity name="departure_evt"/>
      <sensitivity name="bind_evt"/>
      <sensitivity name="unbind_evt"/>
      <sensitivity name="arrival_ntf"/>
      <sensitivity name="departure_ntf"/>
    </process>
    <ports name="dpc"/>
    <ports name="rcfgpc"/>
  </sc_module_controller>
</subModules>
```

Figure 28: A part of the generated SCTLM XMI Model (a part).

A.2. Business Logic Code Generation. At elaboration time, modules, ports, primitive channels, and simulation processes are created and connectivity is established. Thus, SystemC does not support the dynamic creation or modification of the module hierarchy once the elaboration phase has been completed.

When called from the callback end_of_elaboration or during simulation, the sc_spawn function allows the creation of dynamic process instances; these can also be dynamically destroyed during simulation. With dynamic process facility, it would no longer be necessary to preallocate a number of statically defined processes to support the maximum number of possible configurations [44].

It is perfectly appropriate to associate the abstract factory creational pattern with the spawned processes to generate the factory unit of a reconfigurable component. Indeed, abstract factory allows dynamically loading a set of related IPs without explicitly specifying their implementation. Technically, abstract factory is paired with any desired concrete factory to create particular instances at runtime. Different behaviors are then generated by delaying the

Figure 29: Abstract factory metamodel (left) and instantiated FIR filter model (right).

production choice until the instantiation time. By focusing on product interfaces instead of their implementations, applications remain loosely coupled and platform independent.

In order to accurately simulate the FIR filter dynamic reconfiguration, the authors should be able to use the relevant filter factory (32TapFirFactory, 64TapFirFactory, etc.) without knowing which filter implementation (32TapHPF,

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/emf/2002/Ecore','http://www.afactory.org']
[template public generateElement(anEClass : PatternUseCase)]
[comment @main/]
[file (anEClass.name.concat('.cpp'), false, 'UTF-8')]
class [anEClass.AbstractFactory.name/] // abstract factory
{public:
       [for (it : Operation  |anEClass.AbstractFactory.operation )]
              [it.kind/] [it.reurnType/] *[it.name/]()=0;
       [/for]
}
[for (itcf : ConcreteFactory | anEClass.ConcreteFactory)]
class [itcf.name/] : public [anEClass.AbstractFactory.name/]
{public:
[for (itop : Operation | itcf.operation)]
       [itop.reurnType/] *[itop.name/](){ return new [itop.service/](); }
[/for]
}
[/for]
[for (itap : AbstractProduct | anEClass.AbstractProduct)]
class [itap.name/]
{public:
[for (itop : Operation | itap.operation)]
       [itop.kind/] [itop.reurnType/] [itop.name/]()=0;
[/for]
};
[/for]
[for (itcp : ConcreteProduct | anEClass.ConcreteProduct)]
class [itcp.name/] : public [itcp.parent/]
{
[for (itpr : Property | itcp.property)]
       [itpr.type/] [itpr.name/] = [itpr.value/];
[/for]
[for (itop : Operation | itcp.operation)]
       [itop.reurnType/] [itop.name/](){}
[/for]
};
[/for]
[/file]
[/template]
```

Figure 30: Abstract factory template.

64TapHPF, etc.) will be used in the factory unit. The metamodel and a conformed model instance presented in Figure 29 are used as inputs for the transformation engine. Figure 30 depicts the template used for this purpose.

Data Availability

The MARTE4DPR and MARTE4SCTLM profiles, metamodels, and templates used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this study.

References

[1] S. H. Sfar, R. Tourki, and I. Bennour, "Stepwise SystemC/TLM-2 models structuring and optimizations," in *Proceedings of the 11th International Design and Test Symposium (IDT)*, pp. 199–204, IEEE, Hammamet, Tunisia, 2016.

[2] IEEE, 1666-2011, IEEE Standard for Standard SystemC Language Reference Manual, 2012.

[3] M. Jung, C. Weis, and N. Wehn, "DRAMSys: a flexible DRAM subsystem design space exploration framework," *IPSJ Transactions on System LSI Design Methodology*, vol. 8, pp. 63–74, 2015.

[4] A. Ben Ahmed, O. Mosbahi, M. Khalgui, and Z. Li, "Toward a new methodology for an efficient test of reconfigurable hardware systems," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 4, pp. 1864–1882, 2018.

[5] G. Fornari and V. A. de Santiago Junior, "Dynamically reconfigurable systems: a systematic literature review," *Journal of Intelligent and Robotics Systems*, vol. 95, no. 3-4, pp. 829–849, 2019.

[6] N. Fredj, Y. Hadj Kacem, and M. Abida, "A model driven-based approach for managing unanticipated runtime adaptation of RTE systems," in *Proceedings of the 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 317–322, IEEE, Gold Coast, Australia, 2019.

[7] D. de la Fuente, J. Barba, J. Caba, P. Penil, J. C. Lopez, and P. Sanchez, "Building a dynamically reconfigurable system through a high-level development flow," *Languages, Design Methods, and Tools for Electronic System Design, Lecture Notes in Electrical Engineering*, Vol. 385, Springer, Cham, Switzerland, 2016.

[8] OMG, Unified Modeling Language (UML), 2017, https://www.omg.org/spec/UML/About-UML/.

[9] N. Kahani, "AutoModel: a domain-Specific language for automatic modeling of real-time embedded systems," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, pp. 515–517, ACM/IEEE, Gothenburg, Sweden, 2018.

[10] A. Mizzi, J. Ellul, and G. Pace, "D'Artagnan: an embedded DSL framework for distributed embedded systems," in *Proceedings of the Real World Domain Specific Languages Workshop (RWDSL)*, pp. 1–9, Vienna, Austria, 2018.

[11] C. Metrailler and P. A. Mudry, "ESPeciaL: an embedded systems programming language," in *Proceedings of the 6th ACM SIGPLAN Symposium on Scala*, pp. 51–55, ACM, Portland, OR, USA, 2015.

[12] P. Kukkala, J. Riihimaki, M. Hannikainen, T. D. Hamalainen, and K. Kronlof, "UML 2.0 profile for embedded system design," in *Proceedings of the Design, Automation and Test in Europe (DATE'05)*, pp. 710–715, IEEE, Munich, Germany, 2005.

[13] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio, "A SoC design methodology involving a UML 2.0 profile for SystemC," in *Proceedings of the Design, Automation and Test in Europe (DATE'05)*, pp. 704–709, IEEE, Munich, Germany, 2005.

[14] Y. Wang, X. G. Zhou, B. Zhou, L. Liang, and C. L. Peng, "A MDA based SoC modeling approach using UML and SystemC," in *Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT'06)*, p. 6, IEEE, Seoul, Korea, 2006.

[15] B. A. Correa, J. F. Eusse, D. Munera, J. E. Aedo, and J. F. Velez, "High level system-on-chip design using UML and SystemC," in *Proceedings of the 4th Congress of Electronics, Robotics and Automotive Mechanics (CERMA2007)*, pp. 740–745, IEEE, Morelos, Mexico, 2007.

[16] OMG, System Modeling Language (SysML), 2019, https://www.omg.org/spec/SysML/About-SysML/.

[17] OMG, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, 2019, https://www.omg.org/spec/MARTE/About-MARTE/.

[18] X. Peña, F. Rincon, J. Dondo, J. Caba, and J. C. Lopez, "Runtime partial reconfiguration simulation framework based on dynamically loadable components," in *Proceedings of the 11th International Symposium of Applied Reconfigurable Computing (ARC 2015)*, pp. 153–164, Springer, Cham, Germany, 2015.

[19] A. Schallenberg, *Dynamic partial self-reconfiguration: quick modeling, simulation, and synthesis*, Ph.D. Thesis, Department of Computer Science, Oldenburg University, Oldenburg, Germany, 2010.

[20] H. Cerventes and R. S. Hall, "Autonomous adaptation to dynamic availability using a service-oriented component model," in *Proceeding of the 26th International Conference on Software Engineering (ICSE 2004)*, pp. 614–623, Edinburgh, UK, 2004.

[21] J. Estublier and G. Vega, "Reconciling components and services: the apam component-service platform,," in *Proceedings of the IEEE Ninth International Conference on Services Computing*, pp. 683-684, IEEE, Honolulu, HI, USA, 2012.

[22] J. C. Americo and D. Donsez, "Service component architecture extensions for dynamic systems," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC 2012)*, pp. 32–47, Springer, Shanghai, China, 2012.

[23] C. Wang, X. Li, Y. Chen, Y. Zhang, O. Diessel, and X. Zhou, "Service-oriented architecture on FPGA-based MPSoC," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2993–3006, 2017.

[24] R. Carvalho, R. Alencar, and A. Sarmento, "Generation of SystemC simulation models from service level UML diagrams," in *Proceedings of the VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pp. 114–121, IEEE, Salvador, Brazil, 2018.

[25] E. Ebeid, F. Fummi, and D. Quaglia, "HDL code generation from UML/MARTE sequence diagrams for verification and synthesis," *Design Automation for Embedded Systems*, vol. 19, no. 3, pp. 277–299, 2015.

[26] M. Leite and M. A. Wehrmeister, "System-level design based on UML/MARTE for FPGA-based embedded real-time systems," *Design Automation for Embedded Systems*, vol. 20, no. 2, pp. 127–153, 2016.

[27] F. G. C. Ribeiro, A. Rettberg, C. E. Pereira, S. S. C. Botelho, and M. S. Soares, "Guidelines for using MARTE profile packages considering concerns of real-time embedded systems," in *Proceedings of the IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 917–922, IEEE, Emden, Germany, 2017.

[28] A. V. Brito, M. Kuhnle, M. Hubner, J. Becker, and E. U. K. Melcher, "Modelling and simulation of dynamic and partially reconfigurable systems using SystemC," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07)*, pp. 35–40, IEEE, Porto Alegre, Brazil, 2007.

[29] A. V. Brito, G. Silveira, and E. U. K. Melcher, "A methodology for modelling and simulation of dynamic and partially reconfigurable systems," in *Dynamic Modeling*, A. V. Brito, Ed., InTech, London, UK, 2010.

[30] E. Sotiriou-Xanthopoulos, K. Siozios, G. Economakos, and D. Soudris, "A process-based reconfigurable SystemC module for simulation speedup," in *Proceedings of the 2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, pp. 72–79, IEEE, Agios Konstantinos, Greece, 2013.

[31] T. Cervero, J. Dondo, A. Gomez et al., "A resource manager for dynamically reconfigurable FPGA-based embedded systems," in *Proceedings of the 16th Euromicro Conference on Digital System Design (DSD)*, pp. 633–640, IEEE, Santander, Spain, 2013.

[32] E. Suvorova, N. Matveeva, A. Rabin, and V. Rozanov, "System level modeling of dynamic reconfigurable system-on-chip," in *Proceedings of the 17th Conference of Open Innovations Association (FRUCT)*, pp. 222–229, IEEE, Yaroslavl, Russia, 2015.

[33] F. Herrera, I. Ugarte, and E. Villar, "Towards automated implementation of adaptive systems from abstract SystemC specifications," *Design Automation for Embedded Systems*, vol. 16, no. 3, pp. 129–160, 2012.

[34] F. Duhem, F. Muller, and P. Lorenzini, "Methodology for designing partially reconfigurable systems using transaction-level modeling," in *Proceedings of the 2011 IEEE Conference on Design and Architectures for Signal and Image Processing (DASIP)*, p. 7, IEEE, Tampere, Finland, 2011.

[35] F. Duhem, F. Muller, R. Bonamy, and S. Bilavarn, "FoRTReSS: a flow for design space exploration of partially reconfigurable systems," *Design Automation for Embedded Systems*, vol. 19, no. 3, pp. 301–326, 2015.

[36] A. Raabe and A. Felke, "A SYSTEMC language extension for high-level reconfiguration modeling," in *Proceedings of the 2008 Forum on Specification, Verification and Design Languages (FDL 2008)*, pp. 55–60, IEEE, Stuttgart, Germany, 2008.

[37] L. Gong and O. Diessel, "Modeling dynamically reconfigurable systems for simulation-based functional verification," in *Proceedings of the 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 9–16, IEEE, Salt Lake, UT, USA, 2011.

[38] I. R. Qadri, A. Gamatie, S. Meftali, J. L. Dekeyser, H. Yu, and E. Rutten, "Targeting reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation," *International Journal of Embedded Systems*, vol. 4, no. 3-4, pp. 204–224, 2010.

[39] J. Vidal, F. de Lamotte, G. Gogniat, J. P. Diguet, and S. Guillet, "Dynamic applications on reconfigurable systems: from UML model to FPGA implementation," in *Proceedings of the 2011 Design, Automation and Test in Europe (DATE 2011)*, pp. 1–4, IEEE, Grenoble, France, 2011.

[40] S. Cherif, *A model-based approach for the conception of dynamically reconfigurable systems: from MARTE to Reco-MARTE*, Ph.D. Thesis, Lille University of Science and Technology, Villeneuve-d'Ascq, France, 2013.

[41] G. Ochoa-Ruiz, S. Guillet, F. De Lamotte et al., "An MDE approach for rapid prototyping and implementation of dynamic reconfigurable systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 1, pp. 1–25, 2015.

[42] S. Guillet, F. De Lamotte, N. Le Griguer, E. Rutten, G. Gogniat, and J. P. Diguet, "Extending UML/MARTE to support discrete controller synthesis, application to reconfigurable systems-on-chip modeling," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 3, pp. 1–17, 2014.

[43] Eclipse Packages, 2020, https://www.eclipse.org/downloads/packages/.

[44] D. C. Black, J. Donovan, B. Bunton, and A. Keist, *SystemC: From the Ground up*, Springer, Boston, MA, USA, Second edition, 2010.

[45] OSCI, OSCI TLM-2.0 Language Reference Manual, 2009, https://www.accellera.org/downloads/standards.

[46] LocMetrics Tool, 2020, https://www.locmetrics.com.

[47] M. van Amstel, S. Bosems, I. Kurtev, and L. Ferreira Pires, "Performance in model transformations: experiments with ATL and QVT,," in *Proceedings of the 4th International Conference on Theory and Practice of Model Transformations (ICMT'11)*, pp. 198–212, Springer Berlin Heidelberg, Zurich, Switzerland, 2011.

[48] A. Vignaga, "Metrics for measuring atl model transformations," Technical Report, Universidad de Chile, 2009. https://www.dcc.uchile.cl/TR/2009/TR_DCC-20090430-006.pdf.

[49] X. Xu, R. Liu, Z. Wang, Z. Tu, and H. Xu, "RE2SEP: a two-phases pattern-based paradigm for software service engineering," in *Proceedings of the IEEE 13th World Congress on Services*, pp. 67–70, IEEE, Honolulu, HI, USA, 2017.