e-Informatica Software Engineering Journal, Volume 12, Issue 1, 2018, pages: 9–26, DOI 10.5277/e-Inf180101

A Graphical Modelling Editor for STARSoC Design Flow Tool Based on Model Driven Engineering Approach

Elhillali Kerkouche*, El Bay Bourennane**, Allaoua Chaoui***

*Department of Computer Science, Mohamed Seddik Ben Yahia University, Jijel, Algeria

**LE2I Laboratoire, , University of Bourgogne, Dijon, France

***MISC Laboratory, Department of Computer Science and its Applications, Faculty of IT, Abdelhamid Mehri University, Constantine, Algeria

elhillalik@yahoo.fr, ebourenn@u-bourgogne.fr, a_chaoui2001@yahoo.com

Abstract

Background: Due to the increasing complexity of embedded systems, system designers use higher levels of abstraction in order to model and analyse system performances. STARSoC (Synthesis Tool for Adaptive and Reconfigurable System-on-Chip) is a tool for hardware/software co-design and the synthesis of System-on-Chip (SoC) starting from a high level model using the StreamsC textual language. The process behaviour is described in the C syntax language, whereas the architecture is defined with a small set of annotation directives. Therefore, these specifications bring together a large number of details which increase their complexity. However, graphical modelling is better suited for visualizing system architecture.

Objectives: In this paper, the authors propose a graphical modelling editor for STARSoC design tool which allows models to be constructed quickly and legibly. Its intent is to assist designers in building their models in terms of the UML Component-like Diagram, and in the automatic translation of the drawn model into StreamsC specification.

Methods: To achieve this goal, the Model-Driven Engineering (MDE) approach and well-known frameworks and tools on the Eclipse platform were employed.

Conclusion: Our results indicate that the use of the Model-Driven Engineering (MDE) approach reduces the complexity of embedded system design, and it is sufficiently flexible to incorporate new design needs.

Keywords: embedded systems, hardware/software co-design, STARSoC tool, UML, model-driven engineering, Eclipse modelling project

1. Introduction

The increasing complexity of embedded system designs calls for high level specification languages (like StreamsC [1] or others C/C++ based extensions), and for automated transformations towards lower level descriptions. These languages allow to create high level models quickly, run simulations, optimize designs and investigate the efficiency of different algorithms and architectures before generating their corresponding low level implementations. The automatic generation of low level implementation drastically reduces the amount of code to be written by designers, which saves time to market and reduces fabrication costs compared to hand-tuned implementations [2]. For these reasons, the design tools are widely adopted by the embedded system designers' community [3]. The specification of the applications becomes easier at high abstraction levels, since the implementation details are hidden from the designer.

The Synthesis Tool for Adaptive and Reconfigurable System-On-Chip(STARSoC) [4] is one

of those design tools that allow hardware-software co-design, design space exploration and high level synthesis from a StreamsC textual specification. The StreamsC language [5] permits the modelling of the architecture and the behaviour of a complex embedded system containing both Hardware and Software communicating processes. In StreamsC textual models, the architecture of the system is defined with a collection of annotation directives which are used to declare processes and communication between them, whereas processes' behaviours are described in the C programming language. Therefore, these specifications allow for gathering a lot of details (system architecture and processes' behaviours) which increase their length and their complexity, and consequently decrease their legibility.

It is well known that graphical specification is better suited for describing the system components and their relationships, whereas components' behaviours are generally expressed in textual notations (like the C programming language) which allow their reuse as building blocks in new designs. The optimal modelling solution consists in combining textual notations with graphical notations in order to accumulate their advantages. Thereby, every system aspect is provided with the most suitable view (textual/graphical). UML Component Diagrams [6] are widely used to define the structure of a system. A Component Diagram provides a clear view of the organization and the dependency among components in a system, including their contents (source code, binary code or executable) and their interfaces through which they interact with one another. In this work, the Authors propose to develop a graphical modelling editor for the STARSoC design tool. More precisely, it is an approach and a tool support to allow a high-level graphical specification of embedded systems which combines the architectural and behavioural aspects of a system in one model. The architectural aspect is expressed with a UML Component-like Diagram which is an adaptation of the UML Component Diagram to the structural concepts of the StreamsC language, whereas the behavioural aspect is specified in the C programming language. From the graphical specification of a system, this approach permits to automatically generate a clean and correct SteamsC specification. In order to achieve this objective, it is proposed to use the Model-Driven Engineering (MDE) [7] approach which is based on meta-modelling and Model Transformations, and to employ well-known frameworks and tools under the Eclipse platform to in this automatic approach.

The rest of the paper is organized as follows. Section 2 outlines the major related works. In Section 3, some concepts of the StreamsC language are presented. Section 4 presents the STARSoC Tool. In Section 5, an overview of the Eclipse Modelling Project is given. In Section 6, the approach is presented and it is applied on an example in Section 7. The last section concludes the paper and gives some perspectives of this work.

2. Related works

In the literature, several research works have been done on the automatic code generation tools for Multi-Processor Systems-on-Chip (MPSoCs) in order to facilitate and to accelerate the design process.

In addition to STARSoC, there are several code generation tools for MPSoCs which use the textual specification of the whole system as input. From this high level specification containing various system parameters, the tools generate a low level description of the system and perform their functionalities which are necessary in the design process, such as simulation, design space exploration, performance evaluation, etc. For example, xENOC [8] is an automatic environment for hardware/software design of Network-on-Chip (NoC)-based MPSoC architectures. xENoC is based on a tool, called NoCWizard which uses an eXtensible Markup Language (XML) specification (including NoC features, Intellectual Properties (IPs) and mapping) to generate many types of NoC instances by using Verilog language [9]. In addition to NoC instances generation, xNoC also includes an Embedded Message Passing Interface (eMPI) supporting parallel task communication. SystemCoDesigner [10] is another design environment for high-level system modelling and simulation, automatic design space exploration

and automatic hardware/software synthesis from abstract model to final implementation. In SystemCoDesigner, the input model is given using SystemC textual language [11] which describes the structural and behavioural aspects of the system. In addition to academic environments, some commercial design environments support the creation of MPSoCs. The most popular are Altera System on a Programmable Chip (SoPC) [12] and Xilinx Embedded Development Kit (EDK) [13]. In these environments, the hardware part description and the hardware-software integration of the final system are strongly automated using an extensive IP cores library. Although textual notations better describe system parameters and aspects for the design and implementation, these notations increase the complexity of system specifications.

On the other hand, several research works have been proposed to adapt the UML notation to the modelling of embedded systems. The advantage of UML is that it can be extended to any particular domain by defining profiles which introduce additional domain-specific modelling concepts and constraints. In this context, many profiles have been proposed for embedded systems design. The SysML (System Modelling Language) profile [14] reuses a subset of UML notation and provides additional extensions needed in system engineering. It offers graphical modelling support for the specification, analysis, design, verification and validation of complex heterogeneous systems that may combine hardware and software components. The MARTE (Modelling and Analysis of Real-time and Embedded Systems) profile [15] is another UML profile which adds capabilities to UML for the development of Real Time and Embedded Systems (RTES). This extension provides support for specification, design and verification/validation phases. In addition, it defines a common way of modelling both the hardware and software aspects of systems (such as the representation of repetitive structures) in order to improve communication between developers. In order to cope with the design complexities of intensive signal and image data processing applications, the DaRT (Data-parallelism for Real-Time) team [16] of LIFL (the Computer Science Laboratory of Lille University, French) developed a design flow methodology and a tool labelled GASPARD2 [17]. Using a subset of MARTE Profile, GASPARD2 follows the Model Driven Architecture (MDA) [18] principles to describe systems at different level of abstractions. It emphasizes system level co-modelling (hardware and software), simulation, models refinement, automatic code generation and IPs integration. The UML-SystemC profile [19] is proposed to take advantages of both UML and the SystemC language. It captures both the structural and the behavioural features of the SystemC language and allows high level modelling of systems with straightforward translation to the SystemC code. In [20], the authors proposed an UML-based design environment, called Koski, for MPSoCs implementations of wireless sensor network applications. It provides a complete design flow covering the design phases from system level modelling to the FPGA (Field Programmable Gate Array) prototyping. Note that only the relevant profiles have been given here. Many other works which combine the UML modelling with embedded system design flow exist in the literature. However, they rarely cover all design phases from requirement modelling to implementation and validation.

In this work, the Authors intend to introduce a straightforward graphical modelling layer for the STARSoC tool. The proposed graphical modelling editor increases flexibility by integrating the UML notation (UML Component Diagram notation) to the STARSoC input specification language (StreamsC). Furthermore, it takes advantages of the MDE approach to rapidl design systems and integrates new design needs.

3. StreamsC language

The StreamsC language is a parallel programming language following the communicating process model [5]. The language is a small set of directives and library functions callable from a conventional C program. The directives are used to declare three distinguished objects: process, stream or signal, whereas the library functions

```
/// PROCESS_FUN <function_name>
/// IN_STREAM <stream element_data_type> <stream_name>
/// OUT_STREAM <stream element_data_type> <stream_name>
/// IN_SIGNAL <signal element_data_type> <signal_name>
/// OUT_SIGNAL <signal element_data_type> <signal_name>
/// PROCESS_FUN_BODY
. . . . . .
... C code ...
. . . . . .
/// PROCESS_FUN_END
```

Figure 1. Format of the PROCESS FUN directive

```
/// PROCESS <process_name> PROCESS_FUN <process_fun_name> [TYPE [SP | HP]] <on_spec>
```

Figure 2. Format of a process directive

are used to communicate stream data between processes. In the StreamsC programming model a process is an independently executing object with a process body. The process body is written in a subset of C syntax and uses intrinsic functions to perform stream or signal operations. A process may be either software or hardware. All declared processes are initiated when the program begins and runs until their subroutine bodies complete their tasks/functions.

In the following, the directives format is recalled for describing processes, streams and signals that a StreamsC program uses. These directives are embedded in specially formatted blocks. Each directive must be on one line and prepended by "///" followed by a keyword identifying the directive and optional parameter(s) [1].

The first set of directives describes the run function of a process. This is the body of code that gets executed when the associated process is initiated. The PROCESS FUN directive gives a name to the run function, input and output streams and signal parameters, followed by an optional parameter to be passed to the process when it is initiated. After the parameter, the body of the function appears as a normal C code, usually containing variable declarations, stream and/or signal communication, and computation. Finally, a keyword directive is used to mark the end of the run function. The format of the PROCESS FUN directive is shown in Figure 1.

The stream and signal names can be used within stream operations within the body of the process. The data type of stream or signal elements precedes the name of the stream or signal. StreamsC provides predefined unsigned and signed integer data types of stream or signal elements for selected bit lengths ranging from 1 to 64. The supported bit lengths are 1, 2, 4, 6, 8, 12, 16, 18, 20, 24, 32, 40, 48, 64, 128. A simple convention is used to name these predefined types. The signed types have the name **sc_int <bit length>**. The unsigned types have the name **sc_uint<bit length>**.

To describe a process to StreamsC, the PROCESS directive is used. A process has an associated run function and it is an SP (software process) or HP (hardware process) type. If omitted, SP is assumed. Figure 2 shows the format of the PROCESS directive.

The last directive CONNECT is used to connect processes via streams and signals. To connect two processes, the name of one process's stream or signal is associated with the name of another process's stream or signal. In Figure 3, the stream or signal formal parameter defined in the PROCESS FUN directive is generically referred to as a port. The CONNECT directive must be specified from "source" to "destination" (see Figure 3).

Note that the connections between processes must be one-to-one. Broadcast patterns and many-to-one connections are not supported.

An example of the use of these directives to declare and connect processes is shown in Figure 4. There are two software processes called

```
/// CONNECT <process_name>.<port> <process_name>.<port>
      Where: <port> ::= stream or signal name from a PROCESS_FUN directive
```

Figure 3. Format of a StreamsC CONNECT directive

```
//
// Process Functions definitions
//
/// PROCESS_FUN setup_run
/// OUT_STREAM sc_uint4 data
/// PROCESS FUN BODY
    ... C code ...
/// PROCESS_FUN_END
/// PROCESS_FUN finish_run
/// IN_STREAM sc_uint4 processed_data
/// PROCESS_FUN_BODY
    ... C code ...
/// PROCESS_FUN_END
/// PROCESS_FUN p_run
/// IN_STREAM sc_uint4 str1
/// OUT_STREAM sc_uint8 str2
/// PROCESS_FUN_BODY
    ... C code ...
/// PROCESS_FUN_END
//
// Process definitions
//
/// PROCESS setup PROCESS_FUN setup_run
/// PROCESS p_1 PROCESS_FUN p_run TYPE HP
/// PROCESS p_2 PROCESS_FUN p_run TYPE HP
/// PROCESS finish PROCESS_FUN finish_run
//
// Connections
//
/// CONNECT setup.data p_1.str1
/// CONNECT p_1.str2 p_2.str1
/// CONNECT p_2.str2 finish.processed_data
```

Figure 4. CONNECT directives example

setup and finish, and two hardware processes which are instances of the p process. The first instance of the p process (p_1) receives stream data from the setup process. The second instance of the p process (p_2) receives data from the previous instance and outputs data to the finish process.

4. STARSoC design tool

STARSoC [4] is a framework for hardware/software co-design, design space exploration and rapid prototyping on an FPGA (Field Programmable Gate Array) platform for Multi-Processor Systems on Chip (MPSoCs). The overall design flow of the STARSoC tool is summarized in Figure 5.

The design methodology in the STARSoC tool starts from a global model of an application which is a set of communicating processes described in the StreamsC textual language. In the StreamsC model, a process may be either a software process (SP) or a hardware process (HP). Software and hardware processes represent the software and hardware part of the system, respectively. The hardware and software partitions are defined by the user. Note that this design

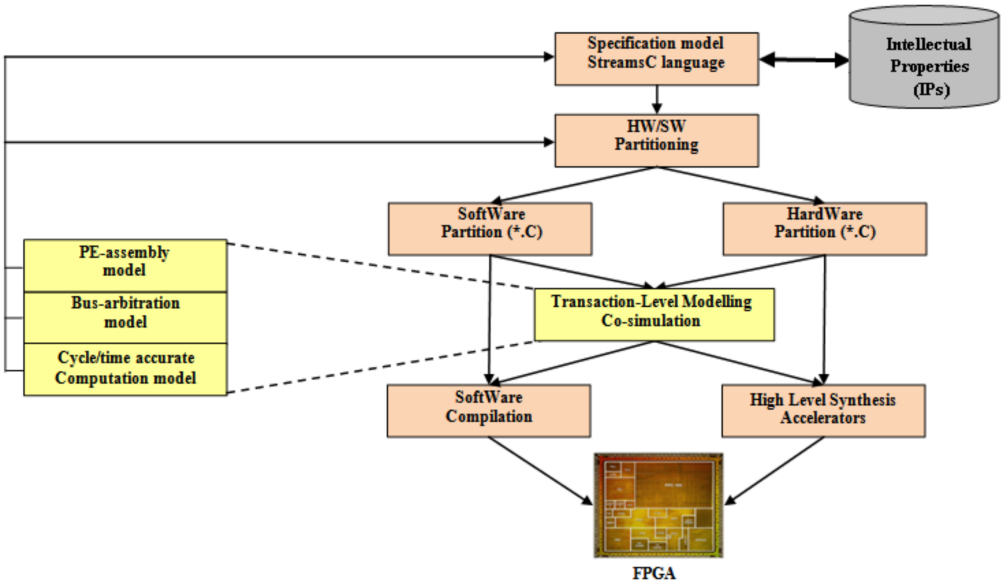


Figure 5. STARSoC design flow [21]

flow is based mainly on reusing open source Intellectual Properties (IPs) for both hardware and software parts.

After hardware-software partitioning, the hardware part is synthesized in Register-Transfer Level (RTL) re-using the StreamsC compiler [22]. In addition, the hardware interface allowing the two partitions, i.e. hardware and software, to communicate is also generated in the RTL code. The obtained RTL code is then downloaded to the FPGA. The software part will be compiled and re-instrumented to generate the machine code of the software processes. This machine code is then downloaded into the program memory of each available processor in the generated MPSoC platform. As a result, STARSoC generates a bus-based MPSoC platform from a high-level application specification.

Before building a prototype for an application, the STARSoC performs a hardware/software co-simulation to validate the behaviour for both hardware and software components and also the interaction between them. In addition, co-simulation permits the performance analysis and rapid exploration of several solutions containing different descriptions of the system components. For this purpose, The STARSoC tool uses Transaction-Level Modelling (TLM) framework [23] which is commonly used for the fast simulation and design exploration of a complex System on Chips (SoCs) at several levels of abstraction and detail. TLM proposes four well-defined transaction level abstraction models that can be independently validated, simulated and estimated. In these models, the application is represented as a set of communicating processes where the communication and the computation are explicitly separated. These processes perform computations and communicate with other processes through an abstract channel.

On the basis of the specification model which describes system functionality without any architecture details (obtained from process codes), the STARSoC tool performs co-simulation by using the following TLM model levels shown in Figure 5:

- *PE-assembly model:* it is made up with multiple processing elements (PEs) connected by channels.
- *Bus-arbitration model:* it represents a refined PE-assembly model in the communication part.
- *Cycle/time-accurate computation model:* It contains cycle accurate computation and approximate-timed communication. This model can be generated from the bus-arbitration model.

The advantage of this approach is that it allows designers to exploit the platform at the earlier stages of the design flow.

5. Eclipse modelling project – overview

The Eclipse Modelling Project [24] is a collection of frameworks and tools for the Model Driven Engineering on the Eclipse platform. In short, they provide a wide range of solutions for various aspects of model driven development, from language definition, generative development of language editors to code generation as well as model verification and validation [25]. In the following, some of the tools from Eclipse Modelling Project that have been used in this work are introduced. These tools are specifically recommended as a basis for developing a graphical editor for the STARSoC tool.

5.1. Eclipse Modelling Framework (EMF)

The Eclipse Modelling Framework [26] forms the basis for all Eclipse Modelling Project tools. It represents the modelling framework and the code generation facility for specifying meta-models and managing model instances. More precisely, EMF includes its own meta-modelling language called Ecore which is used for defining the abstract syntax of modelling languages [27]. From a modelling language specification defined by the Ecore meta-model, EMF generates a simple tree-based editor that enables viewing and editing the instances of the modelling language. In addition, EMF comes with a set of related frameworks for validating models, creating and executing queries against EMF models as well as model transactions.

5.2. Graphical Editing Framework (GEF)

Although EMF is able to generate tree-based editors for model instances of existing meta-models, these editors do not suffice since models are better rendered in a true graphical way. The Graphical Editing Framework [28] provides technology to aid developers in creating rich graphical editors, which are not easily built using native widgets found in the base Eclipse platform. It contains the entire set of tools to define a graphical concrete syntax for each entity of the meta-model according to its appropriate graphical notation. In addition, GEF employs a Model-View-Controller (MVC) architecture which is used to interconnect the graphical part of an editor with the model elements. Thereby, it permits changes to be applied to the model from the view [25]. Although EMF and GEF can be used separately, building a graphical editor requires both of them. In this sense, GEF provides the graphical support required for building a diagram editor on the top of the EMF framework.

5.3. Graphical Modelling Framework (GMF)

The Graphical Modelling Framework [29] provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF. In other words, it provides a generative bridge between the EMF (that allows the meta-model definition) and GEF (a lightweight graphical framework, based on MVC architecture) to help developers creating enhanced graphical editors [25]. Using this framework, one can define graphical notations for existing EMF meta-models.

5.4. Acceleo language

Acceleo is a model-to-text transformation framework that generates text from models [30]. It has been in development since 2006, and was incorporated into the Eclipse M2T project in 2009 [24]. Its purpose is to implement code generators with an easy to use language (according to Object Management Group's MOF model to text transformation language standard [31]) and a good enough tool support (IDE, syntax highlighting, error reporting and debugging features). An Acceleo program requires a meta-model and a model compliant with this meta-model, from which it

generates a text or a code. The meta-model and the model are defined using the EMF framework, which makes Acceleo compatible with other tools based on EMF.

The Acceleo language is a template based approach wherein the text or code to be generated from models are specified as a set of text templates that are parameterized with model elements. More precisely, Acceleo scans the source model according to its meta-model and defines a textual template in the relevant syntax for each text fragment to be generated. The variable parts in the text fragment are specified over model elements. An advantage of this situation is the fact that the structure of the Acceleo templates will directly reflect the structure of the generated text. Thus, the destination text is directly generated, with no need for post-processing. The main feature of Acceleo is that the generated text is mixed with Acceleo syntax.

6. Graphical modelling editor for STARSoC

As it was mentioned earlier, the STARSoC tool starts from a StreamsC textual specification which consists of the architecture and behaviour of a complex embedded system. Gathering all system aspects in StreamsC textual specifications increase their complexity, decrease their readability, and make their understanding and maintenance more difficult. To remedy this, the authors propose to develop a graphical modelling editor for the STARSoC design tool which combines the architectural and behavioural aspects of the system in one model. The architectural aspect is expressed with a UML Component-like Diagram serving this purpose, whereas the behavioural aspect is specified in the C syntax. From this whole model, the StreamsC specification can be generated and all STARSoC design flow activities can be performed.

This section provides the outline of, the process of building the proposed graphical modelling editor using the well-known frameworks defined in MDE approach on the Eclipse platform. The presented approach consists of a process with two steps:

1. The first step consists of specializing UML Component Diagram [6] into StreamsC structural concepts. For this purpose a meta-model for the specialised UML Component Diagram is proposed and a graphical modelling editor is built according to the proposed meta-model.
2. The second step encompasses defining the code generation of StreamsC specification. In order to obtain the automatic and correct process of the code generation, the authors propose to use an Acceleo template language to define and implement the transformation.

6.1. Specializing UML Component Diagram into StreamsC structural concepts

To define a new modelling language or to extend and adapt an existing one, it is necessary to provide an abstract syntax (i.e. a meta-model denoting constructs, their attributes, relationships and constraints) as well as concrete graphical syntax information (the appearance of constructs and relationships in the graphical editor). In this work, the authors prefer to adapt an existing modelling language rather than to develop a new modelling language for specifying systems on the STARSoC tool.

Since StreamsC specification consists of a set of communicating parallel software and hardware processes described with a high level textual language and each process may be linked to a connector by an input port or an output port, the authors propose a modelling language adapted from UML Component Diagram [6] which meets additional needs for specifying embedded systems. UML Component Diagrams are widely used to define the architecture and the structure of a system. A Component Diagram shows components, their contents (source code, binary code or executable one), required interfaces, ports and relationships between them. For this purpose, the authors proposed to meta-model the structural aspect of StreamsC language expressed in the UML Component-like Diagram with the meta-model

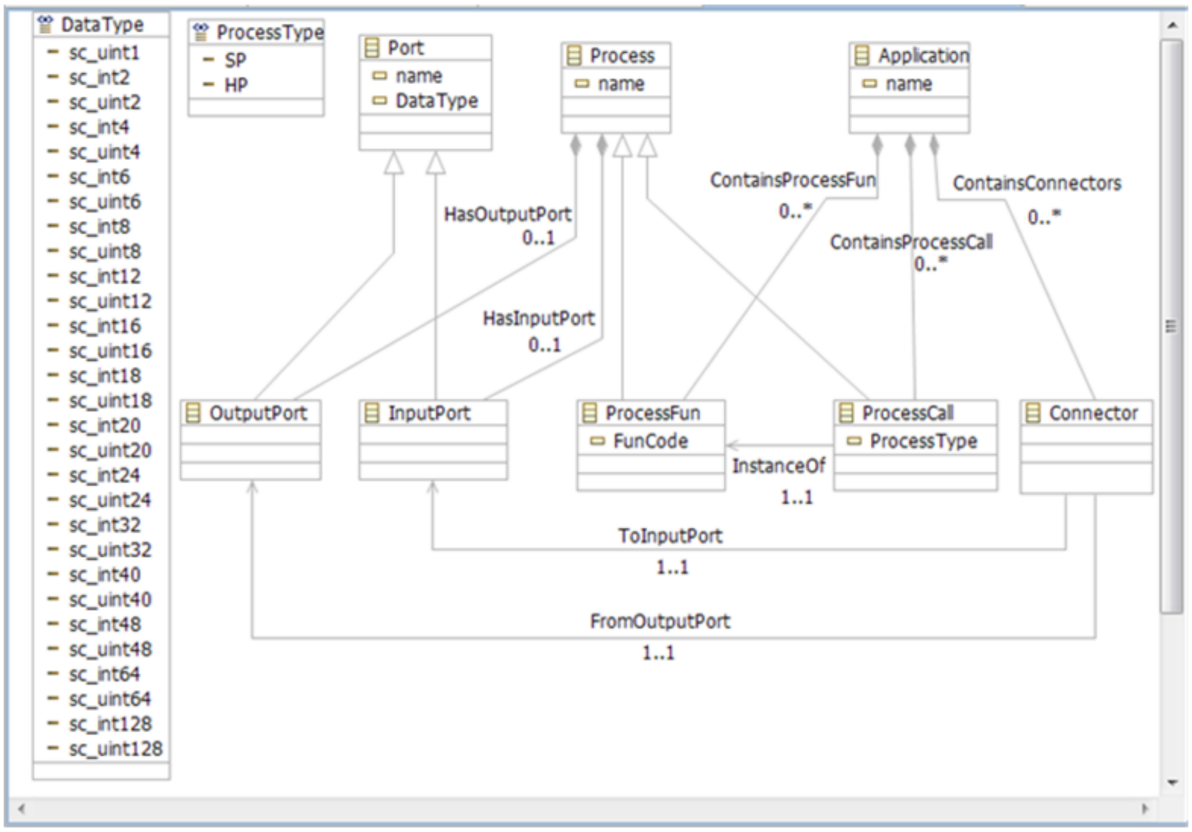


Figure 6. Proposed meta-model in Ecore

shown in Figure 6. In EMF, a meta-model is created and defined in the Ecore format, which is basically a sub-set of UML Class Diagrams. The proposed Ecore model is composed by the following classes:

The *Application* class (attribute name: *name*) represents the application. It contains all the elements used in the application which are process function definitions (*ProcessFun*), process definitions (*ProcessCall*) and connections between processes (*Connector*). The containment relations between the Application class and these elements are specified with Composition relations as shown in Figure 6.

The *ProcessFun* class represents the run functions of processes. It has a String attribute named *FunCode* containing the function code that gets executed when the associated process is initiated.

The *ProcessCall* class represents initiated processes in the application. Each process has an associated run function which is specified with an *Instanceof* association, and a *ProcessType* attribute to indicate the type of the process. The *ProcessType* attribute takes its value from *ProcessType* enumeration class which is *SP* (Software Process) or *HP* (Software Process).

The *Connector* class represents the connections between processes via Ports. A connector has two associations with two other classes called *OutputPort* and *InputPort*, which are sub-classes of the *Port* class.

The *OutputPort* class describes the output ports of source processes, whereas the *InputPort* class represents the input ports of destination processes. *OutputPort* and *InputPort* classes inherit two attributes from the *Port* class: the name of the port and the data type of the stream or signal which takes its value from the *DataType* enumeration class.

In addition, *OutputPort* and *InputPort* classes are contained in the *Process* class which is the abstract class of *ProcessFun* and *ProcessCall* classes.

Despite its expressiveness, Ecore cannot cover all modelling constraints for a modelling language using only graphical elements. Usually, OCL is

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore#/';

package STARSoC : STARSoC = 'http://STARSoC/'
{
    class Application
    {
        attribute name : String[?] = 'MyApplication';
        property ContainsConnectors : Connector[*] { composes };
        property ContainsProcessCall : ProcessCall[*] { composes };
        property ContainsProcessFun : ProcessFun[*] { composes };
    }
    class Connector
    {
        invariant Connector_Must_Connect_Two_Ports_of_The_Same_DataType:
        self.ToInputPort.DataType = self.FromOutputPort.DataType;
        property ToInputPort : InputPort[?];
        property FromOutputPort : OutputPort[?];
    }
    class ProcessCall extends Process
    {
        invariant ProcessCall_InPort_DataType_Is_As_ProcessFun_InPort_DataType:
        self.HasInputPort.DataType = self.InstanceOf.HasInputPort.DataType;
        invariant ProcessCall_outPort_DataType_Is_As_ProcessFun_outPort_DataType:
        self.HasOutputPort.DataType = self.InstanceOf.HasOutputPort.DataType;
        attribute type : ProcessType[?];
        property InstanceOf : ProcessFun[1];
    }
    class ProcessFun extends Process
    class Process
    class Port
    class InputPort extends Port;
    class OutputPort extends Port;
    enum ProcessType
    enum DataType
}
```

Figure 7. Corresponding OCL invariants of the rules

employed to define additional constraints as the so-called well-formedness rules. These rules are implemented in OCL as invariants which are attached to meta-model classes in order to describe properties that should always be satisfied for every model. Thus, the invariant constraints are defined on the meta-model and validated on the model level using the EMF Validation Framework [32]. By introducing the OCL invariants for meta-model classes, a modelling language is more precisely defined leading to models with higher quality.

For this purpose, the proposed Ecore model was enriched with three OCL invariant constraints. These invariants allow the user to check the correctness of the described models with respect to their construction rules as stated in the StreamsC language. In the following part, these rules are described in a natural language, and subsequently the corresponding OCL invariants in the OCLinEcore text editor [33], which embeds the OCL expressions directly into Ecore models by annotating the relevant classes, are shown in Figure 7.

Rule 1: The two end ports of the *Connector* must have the same data type to assure their compatibility.

Rule 2: *ProcessCall* must have the data type of the input port as declared in the input port of its corresponding *ProcessFun*.

Rule 3: *ProcessCall* must have the data type of the output port as declared in the output port of its corresponding *ProcessFun*.

EMF from the proposed Ecore model was used to generate a simple tree-based editor for the modelling language that enables editing and viewing model instances. To develop its graphical modelling editors, both GEF and GMF were used

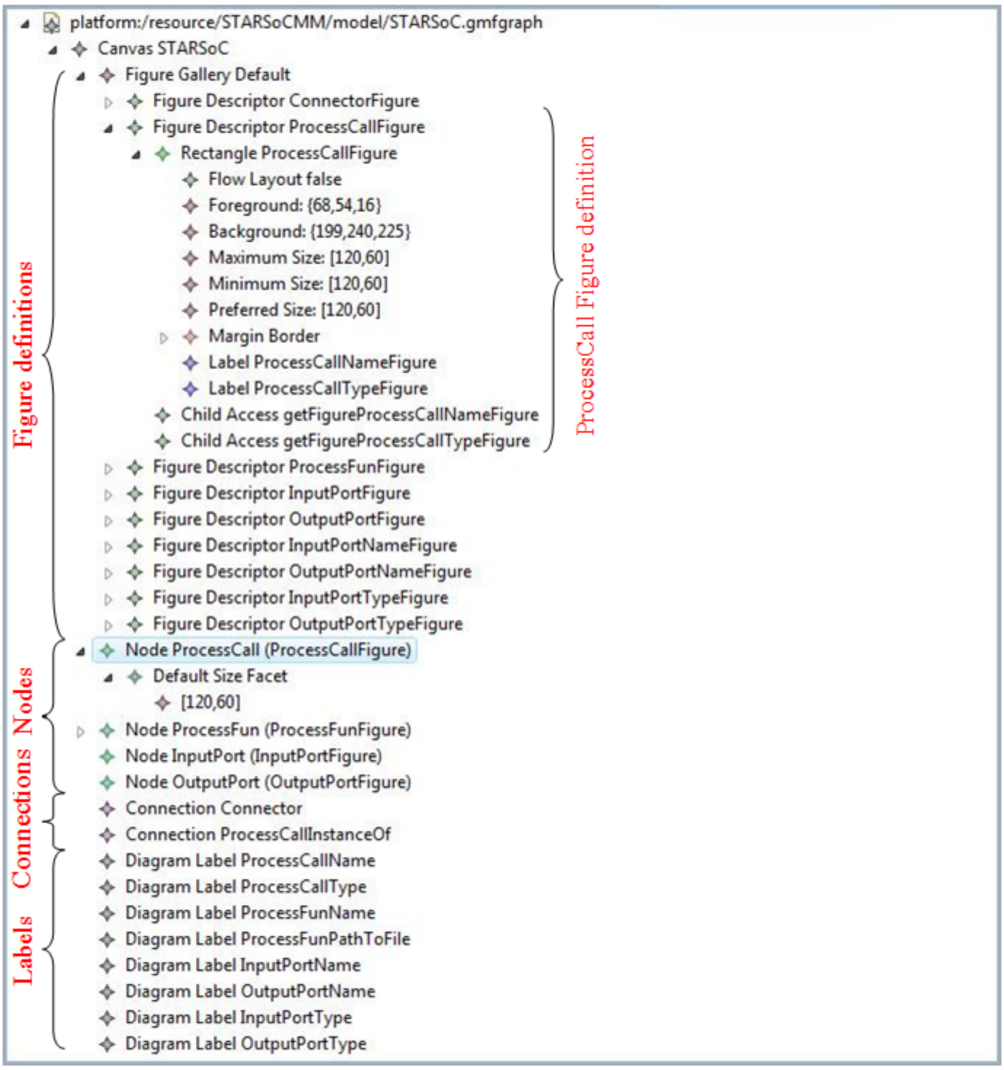


Figure 8. Graphical concrete syntax definition

to define the Graphical model and the Tooling model and Mapping model, respectively.

The Graphical model defines the concrete syntax of the modelling language according to their appropriate graphical notations. It includes information related to the graphical elements (i.e. nodes, labels, connections and decorations for connection ends) that will appear in the editor. The Graphical Model contains also a *Figure Gallery* that contains figures which are used to define shapes. The elements that define the nodes, connections and labels are under the *Figure Gallery* root in the graphical model. Figure 8 shows the graphical definition model for the proposed Ecore model. For example, the *ProcessCall* node uses the rectangle shape defined under *ProcessCallFigure* Figure Descriptor. The rectangle sizes, colours, borders and labels are described separately as rectangle attributes. Similarly, each node element of the Ecore model references the corresponding Figure Descriptor.

The Tooling model defines the toolbar, menus to be used and other periphery to facilitate the management of the model content in the editor. The main focus of the Tooling model is the toolbar definition. The toolbar is defined within a Palette and contains Tool Groups which contain the Tools. In Figure 9, the Tooling definition model for this editor consists of three Tool Groups, namely *Processes*, *Ports* and *Connectors*. The *Processes* Tool Group contains the *ProcessFun* and *ProcessCall* tools for creating the

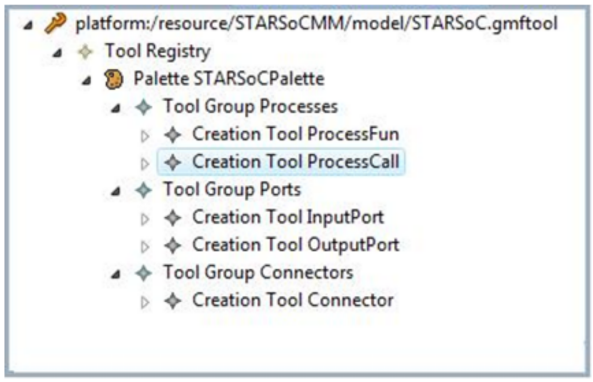


Figure 9. Tooling definition model

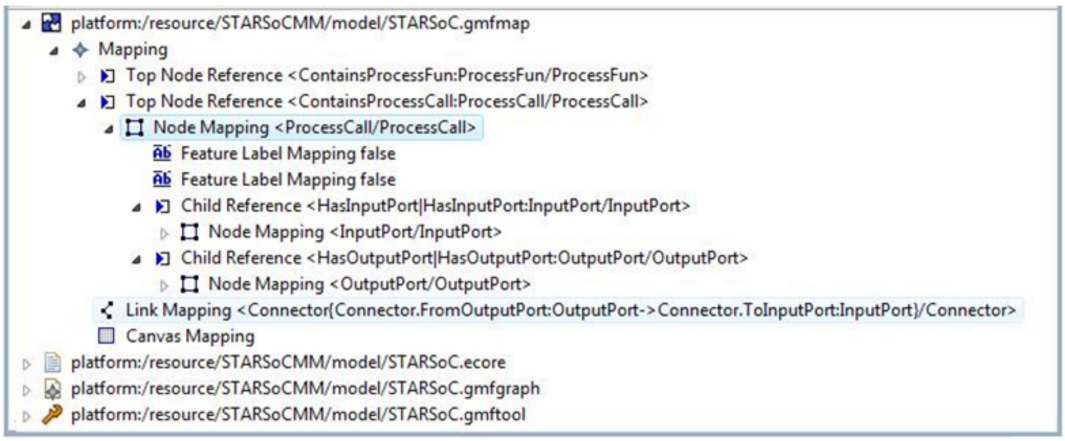


Figure 10. Mapping definition model

ProcessFun and *ProcessCall* elements. The *Ports* Tool Group includes *InputPort* and *OutputPort* tools for creating the Input Port and Output Port elements. The last tool group concerns the creation of *Connectors* in the models.

The Mapping model maps graphical elements from the graphical definition model and creation tools from the tooling definition model to the language constructs from the meta-model. The Mapping model consists of several *Top Node References*, each of which contain one *Node Mapping*. The *Node Mapping* is used to map an element in the graphical model to both the construct in the meta-model and to the creation tool. In addition, it is within the *Node Mapping* that the *Label Mappings* and *Child References* are defined. *Label Mappings* map a *Diagram Label* in the graphical Model to an attribute in the meta-model class that is referenced by the enclosing *Node Mapping*. *Child References* allow meta-model elements to have children, where each child contains an inner *Node Mapping*. In addition to *Top Node References*, *Link Mapping* is used to specify information about a link. It contains information about a source feature, target feature, graphical representation, creation tool, and many other properties. For instance, according to the mapping model in Figure 10, *ProcessCall* elements (Fig. 6) are created by means of the Creation Tool *ProcessCall* (Fig. 9) and the graphical representation for them is the *ProcessCall* Figure definition (Fig. 8). For each *ProcessCall* the corresponding *"name"* and *"ProcessType"* attributes are also visualized because of the specified *Feature Label Mappings* which relate the attribute *"name"* (resp. the attribute *"ProcessType"*) of the *ProcessCall* class with the diagram label *ProcessCallName* (resp. *ProcessCallProcessType*) defined in the graphical definition model.

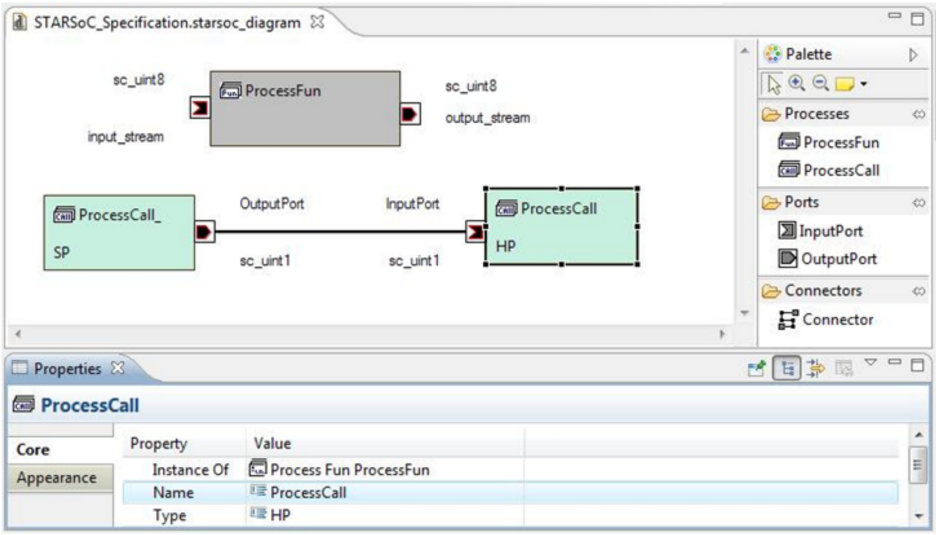


Figure 11. Generated editor for STARSoC

Finally, the Mapping model is transformed into a diagram generator model from which a diagram editor can be generated. Figure 11 shows the graphical modelling editor generated form EMF and GMF models defined for specifying systems on the STARSoC design tool. The editor shows the graphical elements in the diagram and the tools in the palette. Furthermore, GMF provides more advanced features such as annotating, zooming and layouting for the generated editor. The properties of a graphical element can be accessed through the properties view.

6.2. Code generation of StreamsC specification

The next step is the transformation of the graphical specification of a system into its equivalent StreamsC specification using the Acceleo transformation language. In order to do that, the preceding transformation was composed with a set of Acceleo templates (see Figure 12) that traverses the elements of the source model (instances of meta-models) and generates the corresponding StreamsC code.

The first Acceleo template *ToStreamsC (App : Application)* is the main template. It creates the file of the StreamsC specification and takes the only instance of the Application class which contains all model elements as a parameter. Using this parameter (*App*), it scans the contained elements and for each element type produces the corresponding StreamsC code. To achieve this, the *ToStreamsC* template uses three others templates defined for the ProcessFun, ProcessCall and Connector meta-model elements. For example, the template *GenProcessFun(pf : ProcessFun)* takes ProcessFun *pf* as a parameter and writes the run function description of *pf*, which contains the PROCESS_FUN directive, the name of the run function, the input and the output streams, the body of the function and the PROCESS_FUN_END directive, to the output file.

7. Case study

To evaluate the practical usefulness of the proposed graphical editor, a simple application of image processing involving the horizontal edge detection of an image of 256 X 256 pixels coded out of 8 bits was considered. The edge detection is a preliminary step in most image processing techniques. Figure 13 presents the model created in this editor.

The application is defined through two different processes. The first one is a software process, it allows to send the original image, through its output stream, in the direction of the input stream of the second process which is a hardware process. The hardware process performs edge

```
[comment encoding UTF-8 /]
[module generate ('http://STARSoc/')]
[template public ToStreamsC (App : Application)]
   [comment @main/]
   [file (App.name.concat('.sc'), false, 'UTF-8')
   /*______[App.name/].sc_____________________________________________
       Automatically generated streamsc specification
   ____________________________________________________________________*/
   //
   // Process Functions definitions
   //
   [for (processFun : ProcessFun| App.ContainsProcessFun)]
   [GenProcessFun(processFun) /]
   [/for]
   //
   // Process definitions
   //
   [for (processCall : ProcessCall| App.ContainsProcessCall)]
   [GenProcessFun(processCall)/]
   [/for]
   //
   // Connections
   //
   [for (connector : Connector| App.ContainsConnectors)]
   [GenConnector(connector, App)/]
   [/for]
   [/file]
[/template]

[template private GenProcessFun(pf : ProcessFun)]
   /// PROCESS_FUN [pf.name/]
   /// IN_STREAM [pf.HasInputPort.DataType/] [pf.HasInputPort.name/
   /// OUT_STREAM [pf.HasOutput Port.DataType/] [pf.HasOutputPort.name/]
   /// PROCESS_FUN_BODY
   [pf.FunCode /]
   /// PROCESS_FUN_END
[/template]

[template private GenProcessCall(pc : ProcessCall)]
   /// PRoCESS [pc.name/] PROCESS_FUN [pc.InstanceOf.name/] TYPE [pc.ProcessType/]
[/template]

[template private GenConnector(c : Connector, App:Application)]
   /// CONNECT
   [for (pc :ProcessCall| App.ContainsProcessCall)]
      [if(pc.HasOutputPort=c.FromOutputPort)][pc.name/][/if]
   [/for]
   .[c.FromOutputPort.name/]
   [for (pc :ProcessCall| App.ContainsProcessCall)]
      [if (pc.HasInputPort=c.ToInputPort)][pc.name/][/if]
   [/for]
   .[c.ToInputPort.name/]
[/template]
```

Figure 12. Acceleo Templates for StreamsC code generation

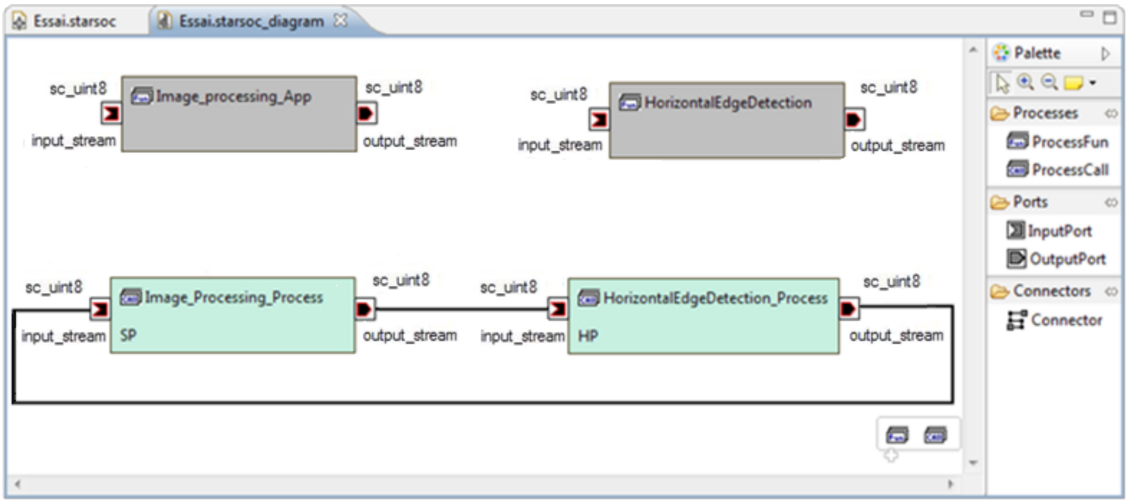


Figure 13. Graphical specification of the application

detection image and returns the resulting image to the software process.

The edge detection algorithm calculates the absolute value of the difference between two consecutive pixels arriving on the data bus of the input streams. The equation of the horizontal edge detection filter is as follows:

$$y(x) = |x(n) - x(n-1)| \tag{1}$$

Only one hardware process is sufficient to perform this calculation. The algorithm of horizontal edge detection is described below (see Figure 14).

Its equivalent StreamsC description is generated from the graphical specification of the application. To generate StreamsC specification in this approach, it is necessary to execute the Acceleo template defined in the previous section. The automatic generated file Essai.sc, which contains the specification, is shown in Figure 15.

This StreamsC specification of the whole application is the basis on which all STARSoC design activities can be performed. Figure 16 shows the development environment for STARSoC tool.

8. Conclusion

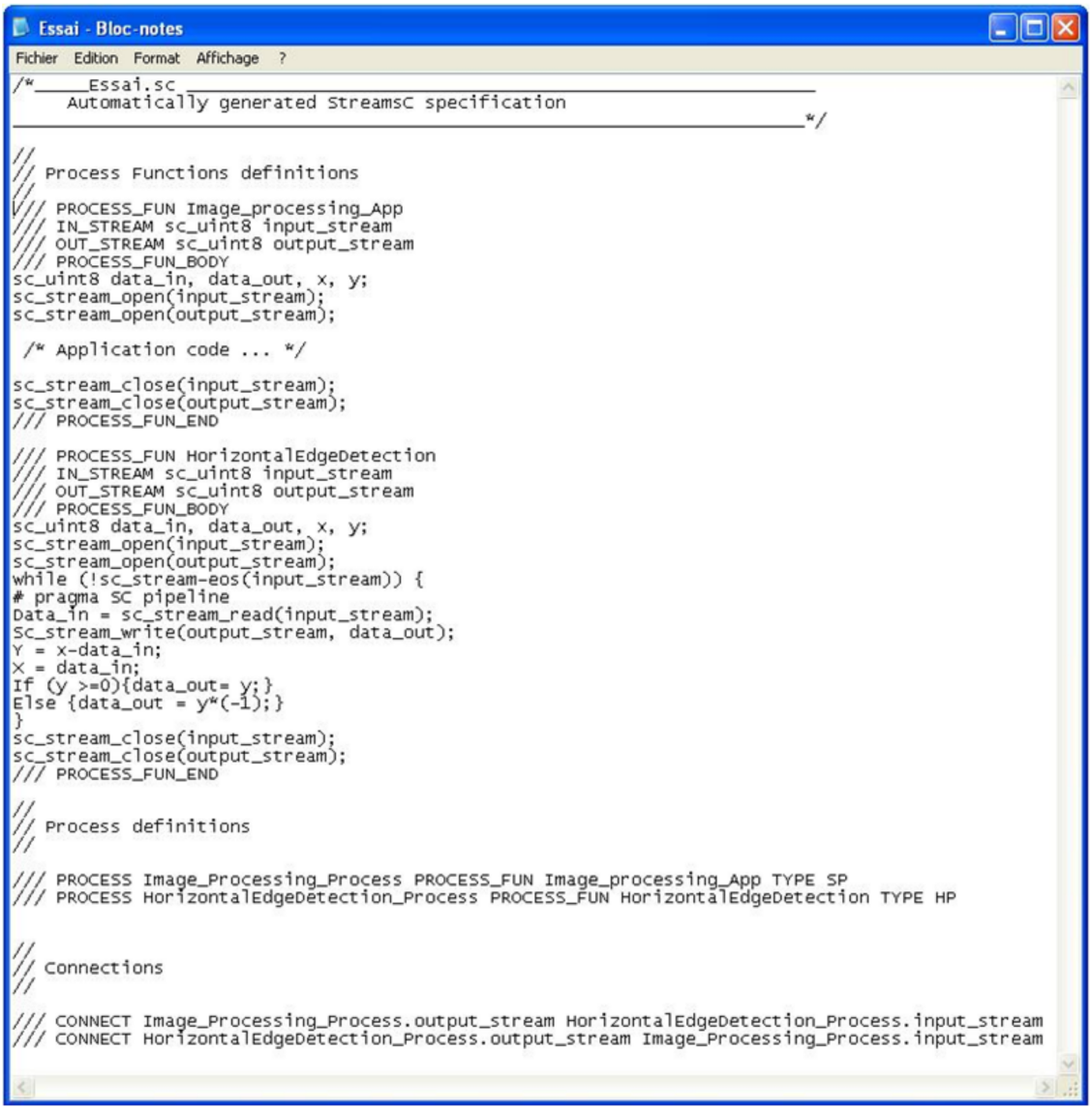
The paper presents some attempts to improve the STARSoC design tool by taking advantage of the Model Driven Engineering techniques. More precisely, Eclipse Modelling Project frameworks and tools (EMF, GEF, GMF, Acceleo,...), which follow the principles of MDE approach, were used to develop a graphical editor for the STARSoC design tool. This editor supports the graphical editing of embedded system models in terms of UML Component-like Diagram and generates the StreamsC textual specifications of these models. The adapted UML Component Diagram is defined in accordance with the embedded system design needs using the Ecore model, whereas the transformation process is defined and executed using the Acceleo framework. The resulting StreamsC specifications are used to perform all STARSoC design tool activities, such as hardware/software co-design, design space exploration and high level synthesis.

According to the authors this approach is sufficiently flexible to incorporate new design needs. Due to the employed Eclipse Modelling Project, revisions of the meta-model almost automatically yield an updated editor and the generation of a text or code is supported as the coding of each meta-model element is analysed separately.

Future work plans encompass the use and adaptation of some UML behavioural diagrams in order to depict the behavioural features of embedded system processes. These behavioural diagrams will be used to automatically generate process codes. One promising direction is to combine existing UML profiles for embedded systems design, such as SysML and MARTE profiles. This

```
sc_uint8 data_in, data_out, x, y;
sc_stream_open(input_stream);
sc_stream_open(output_stream);
while (!sc_stream - eos(input_stream)) {
#pragma SC pipeline
data_in = sc_stream_read(input_stream);
sc_stream_write(output_stream.data_out);
y = x - data_in;
x = data_in;
If (y >= 0) { data_out = y; }
Else { data_out = y * (-1); }
}
sc_stream_close(input_stream);
sc_stream_close(output_stream);
```

Figure 14. Horizontal edges detection algorithm



```
Essai - Bloc-notes
Fichier  Edition  Format  Affichage  ?
/*_____Essai.sc _____________________________________________________
     Automatically generated StreamsC specification
_____________________________________________________________________*/

//
// Process Functions definitions
//
/// PROCESS_FUN Image_processing_App
/// IN_STREAM sc_uint8 input_stream
/// OUT_STREAM sc_uint8 output_stream
/// PROCESS_FUN_BODY
sc_uint8 data_in, data_out, x, y;
sc_stream_open(input_stream);
sc_stream_open(output_stream);

 /* Application code ... */

sc_stream_close(input_stream);
sc_stream_close(output_stream);
/// PROCESS_FUN_END

/// PROCESS_FUN HorizontalEdgeDetection
/// IN_STREAM sc_uint8 input_stream
/// OUT_STREAM sc_uint8 output_stream
/// PROCESS_FUN_BODY
sc_uint8 data_in, data_out, x, y;
sc_stream_open(input_stream);
sc_stream_open(output_stream);
while (!sc_stream-eos(input_stream)) {
# pragma SC pipeline
Data_in = sc_stream_read(input_stream);
Sc_stream_write(output_stream, data_out);
Y = x-data_in;
X = data_in;
If (y >=0){data_out= y;}
Else {data_out = y*(-1);}
}
sc_stream_close(input_stream);
sc_stream_close(output_stream);
/// PROCESS_FUN_END

//
// Process definitions
//

/// PROCESS Image_Processing_Process PROCESS_FUN Image_processing_App TYPE SP
/// PROCESS HorizontalEdgeDetection_Process PROCESS_FUN HorizontalEdgeDetection TYPE HP


//
// Connections
//

/// CONNECT Image_Processing_Process.output_stream HorizontalEdgeDetection_Process.input_stream
/// CONNECT HorizontalEdgeDetection_Process.output_stream Image_Processing_Process.input_stream
```

Figure 15. Generated StreamsC specification

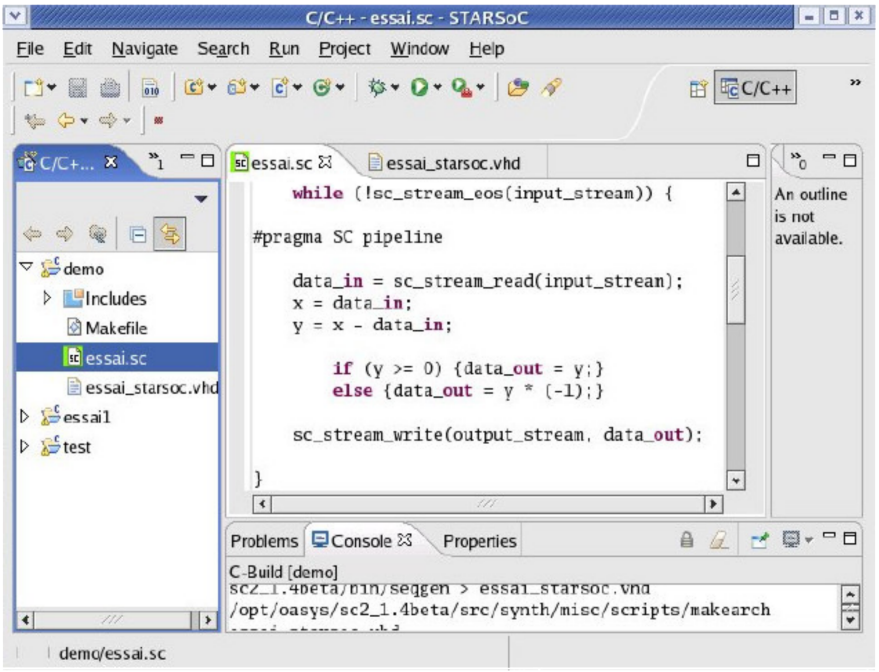


Figure 16. The development environment for STARSoC tool

combination is possible since most of the profiles are focused on the process paradigm.

References

[1] M. Gokhale, *sc2 Reference Manual*, Los Alamos National Laboratory, Los Alamos, NM, USA, 2003.

[2] W. Meeus, K.V. Beeck, T. Goedemé, J. Meel, and D. Stroobandt, "An overview of today's high-level synthesis tools," *Design Automation for Embedded Systems*, Vol. 16, No. 3, Aug. 2012, pp. 31–51.

[3] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K.A. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 30, No. 4, Apr. 2011, pp. 473–491.

[4] A. Samahi and E. Bourennane, "Automated integration and communication synthesis of reconfigurable MPSoC platform," in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, University of Edinburgh. Scotland, United Kingdom: IEEE Computer Society, Aug. 2007, pp. 379–385.

[5] M. B.Gokhale, J.M. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented FPGA computing in the Streams-C high level language," in *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa Valley, CA, USA: IEEE Computer Society, Apr. 2000, pp. 49–56.

[6] *Unified Modeling Language, Version 2.5*, Object Management Group, 2015, OMG Document Number: formal/15-03-01. [Online]. http://www.omg.org/spec/UML/2.5/PDF

[7] A.R. da Silva, "Model-driven engineering," *Computer Languages, Systems and Structures*, Vol. 43, No. C, Oct. 2015, pp. 139–155.

[8] J. Joven, O. Font-Bach, D. Castells-Rufas, R. Martínez, L. Terés, and J. Carrabina, "xENoC – an experimental network-on-chip environment for parallel distributed computing on NoC-based MPSoC architectures," in *16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. Toulouse, France: IEEE Computer Society, Feb. 2008, pp. 141–148.

[9] D. Thomas and P. Moorby, *The Verilog Hardware Description Language*, 3rd ed. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

[10] J. Keinert, M. Streubuhr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, "SystemCoDesigner – an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 14, No. 1, Jan. 2009, pp. 1–23.

[11] T. Grotker, *System Design with SystemC*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[12] *SOPC Builder User Guide, Version 1.0*, Altera Corporation, San Jose, CA, USA, Dec. 2010, Document Number: UG-01096. [Online]. http://www.altera.com/literature/ug/ug_SOPC_builder.pdf

[13] *EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design, Version 13.2*, Xilinx Online Documents, Jul. 2011, OMG Document Number: UG683. [Online]. http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/edk_ctt.pdf

[14] *Systems Modeling Language (OMG SysML), Version 1.4*, Object Management Group, Sep. 2015, OMG Document Number: formal/2015-06-03. [Online]. http://www.omg.org/spec/SysML/1.4/

[15] *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Version Beta 2*, Object Management Group, Jun. 2008, OMG Document Number: ptc/2008-06-09. [Online]. http://www.omg.org/omgmarte/Documents/Specifications/08-06-09.pdf

[16] DaRT team: Dataparallelism for real-time. [Online]. http://www.inria.fr/en/teams/dart/ [Accessed 2016].

[17] GASPARD2 SoC framework. [Online]. http://www.gaspard2.org/ [Accessed 2016].

[18] *Model Driven Architecture Guide, Version 1.0*, Object Management Group, 2003, OMG Document Number: omg/2003-05-01. [Online]. http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf

[19] *UML Profile for System on a Chip (SoC), Version 1.0.1*, Object Management Group, Aug. 2006, OMG Document Number: formal/2006-08-01. [Online]. http://www.omg.org/spec/SoCP/1.0.1/PDF

[20] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T.D. Hämäläinen, J. Riihimäki, and K. Kuusilinna, "UML-based multiprocessor SoC design framework," *ACM Transactions on Embedded Computing Systems*, Vol. 5, No. 2, May 2006, pp. 281–320.

[21] S. Boukhechem and E. Bourennane, "SystemC transaction-level modeling of an MPSoC platform based on an open source ISS by using interprocess communication," *International Journal of Reconfigurable Computing*, Vol. 2008, Sep. 2008, pp. 1–10.

[22] J. Frigo, *sc2 Hardware Library Reference Manual*, Los Alamos National Laboratory, Los Alamos, NM, USA, 2000.

[23] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. Newport Beach, CA, USA: ACM, Oct. 2003, pp. 19–24.

[24] Eclipse modelling project (EMP). [Online]. http://www.eclipse.org/modeling/ [Accessed 2016].

[25] R.C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, 1st ed. Addison-Wesley Professional, 2009.

[26] Eclipse modelling framework (EMF). [Online]. https://eclipse.org/modeling/emf/ [Accessed 2016].

[27] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[28] Graphical editing framework (GEF). [Online]. http://www.eclipse.org/gef/ [Accessed 2016].

[29] Graphical modelling framework (GMF)). [Online]. http://www.eclipse.org/modeling/gmp/ [Accessed 2016].

[30] *User Guide, Version 3.1.0*, The Eclipse Foundation, 2011. [Online]. http://www.eclipse.org/acceleo/support/

[31] *MOF Model to Text Transformation Language, Version 1.0*, Object Management Group, Jun. 2008, OMG Document Number: formal/2008-01-16. [Online]. http://www.omg.org/spec/MOFM2T/

[32] The EMF validation framework project (EMF-VF). [Online]. http://www.eclipse.org/modeling/emf/?project=validation [Accessed 2016].

[33] OCLinEcore editor. [Online]. https://wiki.eclipse.org/MDT/OCLinEcore [Accessed 2016].