DesyreML: a SysML profile for heterogeneous embedded systems*

Alberto Ferrari
ALES S.r.l.
Via Barberini, 50, Roma
alberto.ferrari@ales.eu.com

Leonardo Mangeruca
ALES S.r.l.
Via Barberini, 50, Roma
leonardo.mangeruca@ales.eu.com

Orlando Ferrante
ALES S.r.l. & "Sapienza" University of Rome
Via Barberini, 50, Roma
orlando.ferrante@ales.eu.com

Alessandro Mignogna
ALES S.r.l. & Scuola Superiore Sant'Anna
Via Barberini, 50, Roma
alessandro.mignogna@ales.eu.com

ABSTRACT

We propose a novel language for the formal description of heterogeneous embedded systems (DesyreML). As the main contribution, the language is formally described in terms of semantics and concrete syntax based on the SysML language. We define the concept of thick connector to allow for heterogeneous components communication and computation for multiple semantic domains (synchronous reactive, continuous time, discrete time, discrete-event). As technological application, a verification flow based on model-transformation techniques is described showing the use of an enriched version of the SystemC-AMS simulation kernel that is capable of simulating heterogeneous systems containing combinatorial loops. Finally, the language and the analysis flow are applied to a cruise control case study.

Index Terms— Embedded systems, heterogeneous, language, SysML, SystemC-AMS

1. INTRODUCTION

Embedded systems have become of common use in our daily life. The trend shows an ever-increasing complexity of such system that most of the time should guarantee high performances, safety properties, low power consumption and low costs. The design of new embedded systems requires the integration of more components in a single chip and the interaction of several devices located in different places in the space. Often, the embedded system architectures include a wide variety of heterogeneous components: processors, application specific hardware, DSPs, sensors, actuators, etc. Additionally, a large number of actors are usually involved during the different phases of the design process. Teams, spread all around the world, contribute to the overall design, each one facing a particular design problem and therefore using specific design techniques and specific tools to solve it. The final design result in a composition of heterogeneous modules based on different Model of Computation (MoC) and characterized by aperiodic and periodic computation, event-triggered and time-triggered communication and so on. As a consequence, the capability to support heterogeneity is necessary to deal with the design of such systems. During the entire design process, and especially during the very first development steps, the heterogeneity nature of components should be considered. During the last 10 years, different methodologies, frameworks and tools have been proposed to help the designer during the entire design process. However, there is still the lack of a unique integration framework that would be able to correctly compose models based on different MoCs and to perform some analysis on the resulting system. What is required is a standard methodology to provide interoperability between models of different nature and to cover the whole design flow, from systems requirements to system implementation.
The paper is structured as follows: first a brief description of related works and contributions is reported in section 2. The syntax and semantics of the language are described in section 3, a case study is presented to show how to use the language to face a realistic design problem. Section 4 explains the analysis flow while the simulation backend is reported in section 5. Finally, section 6 concludes the paper.

2. RELATED WORK AND CONTRIBUTIONS

The problem of formally capturing the structure and behavior of heterogeneous systems has been already addressed by several authors. The tagged signal model approach proposes a theoretical framework for comparing properties of different models of computation (MoCs) using a denotational framework [1]. Based on this approach, several other solutions have been proposed to specialize the framework for an important subset of MoCs [2]. Different specification languages and analyses frameworks have been developed to allow designers capturing heterogeneous systems. The PtolemyII and the Metropolis frameworks are modeling and simulation environments based on the tagged signal model theory [3],[4]. The SPEEDS HRC language provides a common semantics and syntax to allow heterogeneous components hosted-simulation [5][6][7]. The MARTE UML profile constraints the semantics of the UML language providing a well-defined notion of time and supporting the specification of components exposing different MoCs [8]. Other approaches uses the SystemC modeling language as glue language for the coordination and execution of heterogeneous components both using interface elements bridging components exposing different

MoCs [9] and extending the SystemC simulation capabilities to capture heterogeneous specification [10]. Most of the above approaches aim at providing a modeling and/or simulation environment for the specification and analysis of embedded systems. Exceptions are the tagged signal model, which provides a denotational framework for the definition of MoCs, rather than models, and the SPEEDS environment that defines a language and protocol to exchange models between different tools.

Our approach is based on the following pillars: 1) a model integration language that supports multiple models of computation, also within the same model; 2) a denotational semantics for the definition of different models of computation and their integration; 3) an operational semantics for the integration of executable models for analysis purposes. The integration language, denotational and operational semantics are connected through the concept of "tag domain" and "tag domain constraints", introduced in section 3.

The focus of the present paper is to describe the model integration language and a simulation framework to demonstrate how the language is connected to analysis. We also provide a synthetic view of the denotational and operational semantics. The integration language, called DesyreML, is an extension of the SysML language. Each component is described in terms of its interface, which is enriched with MoC information. The specification of component behavior is supported in three modalities: clear-box behavior expressed in the SysML language, white-box behavior expressed using external languages such as Simulink, Modelica, etc., and black-box behavior given as executable representations of the component in C/C++ language or in binary form. The integration between different MoCs is specified using special connectors called thick connectors. Our approach places emphasis on the integration capabilities of the language more than on the capability of capturing a super-set of common semantics as already done in previous work. Moreover, we describe a simulation backend for heterogeneous systems based on the DESYREII simulation engine to which a DesyreML model is mapped using a model transformation process.

3. DESYREML PROJECT AND LANGUAGE DESCRIPTION

3.1 DesyreML Language semantics

The semantics of the DesyreML language is structured in two parts, a *denotational semantics* aimed at providing formal underpinning to the language and an *operational semantics* that is defined to provide cross-tool and cross-language integration capabilities at the analysis level.

3.1.1 Denotational semantics

The denotational semantics is defined based on a refinement of the Tagged Signal Model (TSM). Let K denote a set of *tag domains*. Each tag domain $D \in K$ is defined over a set of tag values T_D, for example the set of real numbers, the set of natural numbers, etc. A tag is defined as a function τ: $K \rightarrow \bigcup_{D \in K} T_D \cup \{\perp\}$, such that $\tau(D) \in T_D \cup \{\perp\}$, where the symbol $\perp$ denotes absence of value. Let T denote the set of such tags. An event is defined as a pair $(\tau,v) \in T \times V = E$, where v is an element from a value set V. A *signal* is defined as a subset of events, $s \subseteq E$, such that $(\tau_1,v_1), (\tau_2,v_2) \in s \Rightarrow \forall D \in K, \tau_1(D) = \perp \Leftrightarrow \tau_2(D) = \perp$. In other words, all events of a signal are defined over the same tag domains. Let Π denote a set of *ports*. A behavior is a function σ: $\Pi \rightarrow 2^E$, that assigns a signal to each port. We define a composition operator $\|$ over sets of behaviors: let Π_1 and Π_2 be two sets of ports, E_1 and E_2 two sets of events, $\Sigma_1 \subseteq \{\sigma_1: \Pi_1 \rightarrow 2^{E_1}\}$ and $\Sigma_2 \subseteq \{\sigma_2: \Pi_2 \rightarrow 2^{E_2}\}$ corresponding sets of behaviors. We define $\Sigma_1 \| \Sigma_2 = \{\sigma: \Pi_1 \cup \Pi_2 \rightarrow 2^{E_1 \cup E_2} \mid \sigma|_{\Pi 1} \in \Sigma_1$ and $\sigma|_{\Pi 2} \in \Sigma_2\}$, where the symbol $\sigma|_{\Pi 1}$ means the function σ restricted to Π_1. A *process* is defined as a subset of behaviors. The composition of two processes is defined as the composition of the corresponding subsets of behaviors.

3.1.2 Operational semantics

Fig.1 shows the four layers of the operational semantics: 1) the system specification layer; 2) the domain specific layer; 3) the cross-domain resolution layer; 4) the non-determinism resolution layer. The first layer defines the components that compose the system representation. Components can be homogeneous or heterogeneous. Homogeneous components are specified according their own MoC and their operational semantics is defined by the corresponding MoC specification. Interactions between homogeneous components within the same MoC are resolved in the domain specific layer. Heterogeneous components can be specified over multiple MoCs and can be defined according to their operational semantics, as well as the cross-domain operational semantics. The cross-domain layer provides primitives to specify static and dynamic constraints over tag domains and serves as a layer for ordering and synchronizing the different tag domains, in a similar fashion as MARTE does with clock constraints. The non-determinism resolution layer is used to resolve the behavioral non-determinism required to achieve a deterministic simulation. This layer may use different strategies according to user's needs. Further details on the denotational and operational semantics are outside the scope of this paper, which is focused on the integration language.

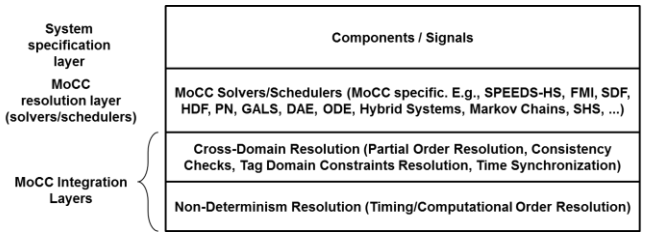

Figure 1 - Operational semantics structure

3.2 DesyreML Language syntax

The DesyreML language syntax has been defined exploiting the extensibility natively offered by the UML and SysML languages with the profile mechanism [11]. SysML is the OMG System Modeling Language and it represents the standard *de facto* for the modeling of complex systems architecture in both academic and industrial projects. A profile is a lightweight extension of the language that allows specializing its syntax using stereotypes that represents both a well-defined syntactic element and a set of additional semantic constraints for each stereotyped metaclass. There are several advantages in using stereotypes for the design of the DesyreML language. First, the lightweight nature of the profile allows enriching the semantics of the language without modifying its basic semantics tenets and this permits the language to be easily accepted by the SysML designers' community. Second, the profile can be encoded using the OMG standard interchange format (XMI) and the modeling tools that support the standard are immediately capable of importing the profile and applying it to existing SysML models. Using this mechanism, the DesyreML profile allows the specification of different semantic domains and gives to the designer the capability of declaring the semantic domain of a precise subset of model elements. The following subsections provide a brief review of the main syntactic elements introduced by the profile.

3.2.1 Semantic domains

MoC, called semantic domains in the DesyreML language, are defined as first class model elements. Each of them may need parameters to be specified to be completely determinate (as the period for the periodic discrete time semantic domain). The profile allows the designer to identify different instances of semantic domains with their specific parameter values. We decided to define the *DESYREML::SemanticDomain* stereotype to specify an abstract semantic domain extending the SysML Block metaclass. Each semantic domain has been modeled using a specific stereotype that inherits from the abstract one. Among the different stereotypes we cite the *DESYREML::ContinuousTimeDomain* and the *DESYREML::DiscreteTimeDomain* that allow the description of continuous and periodic discrete time domains, respectively, and the *DESYREML::DiscreteEventDomain* for the description of the discrete event model of computation (MoC). An additional set of stereotypes (*DESYREML*::CTDVDomain and *DESYREML*::SRDomain) are used to identify the synchronous model of computation defined by the synchronous HRC language.

3.2.2 DesyreML blocks and DesyreML system

In a *DesyreML* design there is exactly one block declared as a root component using the stereotype *DESYREML::System* which may contain one or more blocks (tagged by the *DESYREML::Block* stereotype). A DesyreML block may contain a reference to a specific semantic domain, constraining the semantics supported by its part or may not specify this information, declaring itself to be multi-domain. In the latter case, the resolution of the semantic domain of the component's parts is delegated to the composing blocks.

3.2.3 DesyreML flows and DesyreML thick connectors

The language supports the communication between block instances (parts) by specializing the SysML flow ports. A DesyreML flow port (stereotype *DESYREML::Flow*) is an extension of the SysML flow port which contains information about the semantic domain it supports. DesyreML flows may inherits the semantic domain of the owner block or explicitly declare a specific semantic domain. This approach allows the modeling of heterogeneous components that may support different models of computation for different communication end points. A connection between flows supporting the same semantic domain is, intuitively speaking, equivalent to a classical SysML connector, whereas in case the connector is relating flows supporting different semantics an adaptation mechanism is needed. To explicitly declare this mechanism we introduced the thick connector concept. A thick connector is a profile of the SysML connector (*DESYREML::ThickConnector*) that is used to identify the need of an adapting mechanism between flow ports that support different MoCs. The DesyreML profile provides a predefined list of thick connectors covering a set of adaption layers. Nevertheless, custom thick connectors can be easily introduced using the profile mechanism. As an example of predefined connectors, consider the *DESYREML::DT2CTThickConnector* and *DESYREML::CT2DTThickConnector* used to adapt the continuous and the discrete time MoCs. Each connector may provide a set of parameters to the designer to better specify the adapting mechanism. For example for the *DT2CTConnector* the interpolation method (zero-hold, linear, etc.) is aconnector's parameter.

3.3 Cruise Control case study

We consider as a case study the design of a cruise control system represented by a DesyreML system. The structure of the system has been captured using a block definition

diagram and the connections between flow ports have been described in an internal block diagram (Fig. 2).

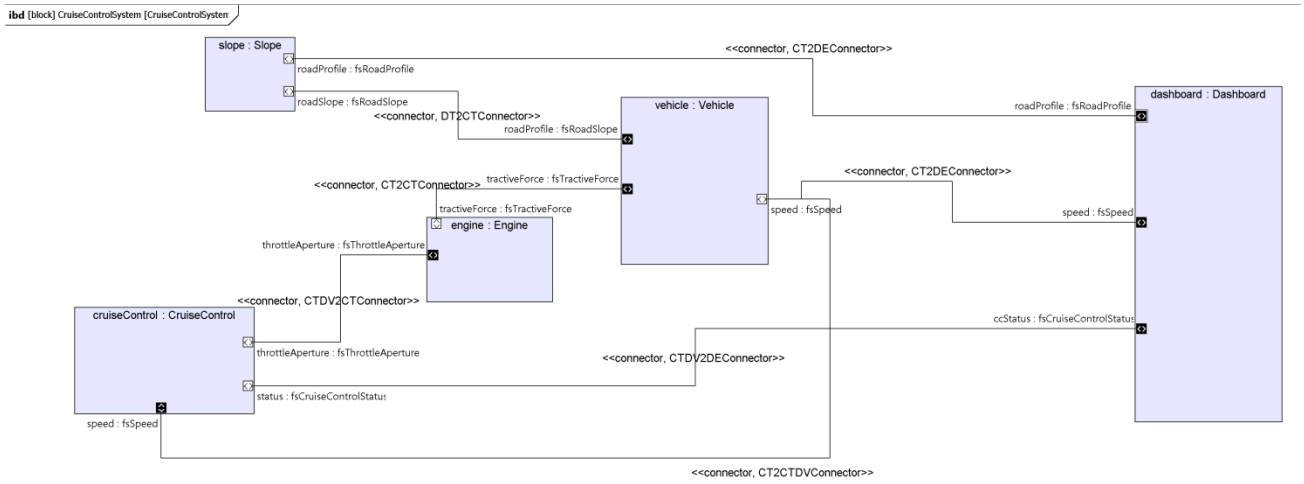

Figure 2 – Cruise Control Internal Block Diagram

The cruise control system is decomposed into five blocks. The *Slope* block models a sensor of the road profile and slope, the *EngineCar* block represents the dynamics of the engine, the *Vehicle* block captures the dynamics of the vehicle, the *ECU* block represents the main controller and the *Dashboard* block represents the visual panel of the system. Table 1 summarizes the semantic domains of each block/flow port.

Table 1: Semantic domains

Block	Flow port	Semantic domain
Slope	road_profile	Continuous Time
	road_slope	Discrete Time
ECU	any	Synchronous (HRC)
EngineCar	any	Continuous Time
Dashboard	Any	Discrete Event
Vehicle	Any	Continuous Time

4. ANALYSIS FLOW

The DesyreML profile has been used as input language for an analysis flow based on a verification environment for heterogeneous systems. The entire analysis flow is depicted in Fig. 3. As a starting point a model of the system is described in SysML using the DesyreML profile defining the semantic domains of each block and flow port. The model is then elaborated using a model transformation step producing an intermediate model which can be used to generate an executable representation of the input model based on a simulation framework called DESYREII. Finally, a simulator is automatically built and run. DESYREII is a distributed embedded system simulator for performance analysis and verification developed by A.L.E.S. S.r.l. [12]. DESYREII is based on SystemC and integrates different technologies such as SPIRIT IP-XACT schema, SPEEDS HRC Metamodels. It provides the capability of importing, MATLAB Simulink and SystemC-AMS models. A DESYRE simulator is described adopting the Platform Based Design (PDB) methodology consisting of a layered structure where each layer provides services to the upper layer and relays on services offered by the lower layers. The framework provides a set of libraries and IPs to model system communication (busses, controllers, protocols stack), system computation (RTOS) and function-to-architecture mapping. Application functionalities can be directly defined by the user, using C++/SystemC (referred to as black-box components) or can be imported from other tools such as MATLAB Simulink ® (referred to as white-box components), using dedicated import flows. The structure of the system to simulate is completely described using IP-XACT compliant XML files. Those file are then passed to the DESYREII model builder that instantiates, interconnects and configures the required IPs, without the need of re-compiling any component. Once the entire netlist has been instantiated, the DESYRE core invokes the SystemC kernel to simulate the system.

DESYRE supports a model-based representation of systems using an internal meta-model. A system is composed by components exposing a well-defined interface in terms of interaction points. Each component is part of a library of model elements and may have attached an executable behavior which should be compliant with the DESYRE simulation protocol in order to be imported in the simulator and that can be part of a library of model elements.

4.1 Model transformation

The transformation process has two objectives. On one hand it produces a structurally equivalent DESYRE representation of the input model. On the other hand it introduces components in the target model in order to

support the semantic adaptation specified in the thick connectors and needed to coordinate heterogeneous components.

The technology used to perform the model transformation step is an internally developed Java embodiment of the OMG Query/View/Transformation (QVT) language called JQVT [13]. The JQVT library aims at providing an industry-level operational implementation of the QVT language. It supports the definition of QVT mappings and the definition of mappings inheritance, disjunction and merging. JQVT allows capturing the mapping relation that links a source model element to a target model element and it supports the *resolve* and *resolveIn* operators to retrieve the set of mapping source model elements from a given mapped target model element. JQVT does not support the entire QVT specification. However, it has been extensively used as translation infrastructure of different tools for the translation of industry-level sized models.

Continuing with the description of the analysis flow, after the target model is generated, a model to text process produces the simulator artifacts. In particular, the file system structure of the simulator is build and for each component a suitable DESYRE wrapper is generated to import the behavior of the component provided as executable artifact (C/C++ code or dll files). Finally an IP-XACT representation of the interconnection is automatically generated as well as the support files for the simulator compilation and linking processes.

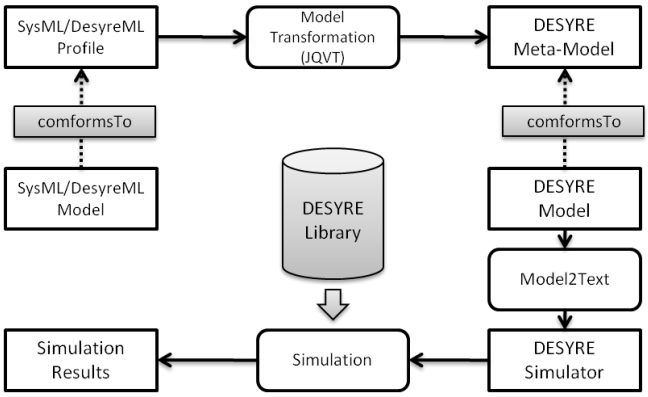

Figure 3 – Analysis Flow

4.2 Application to cruise control case study

To better describe the transformation process, we consider its execution to the cruise control case study. Each SysML block has been translated to a DESYRE component and each port's flow to a DESYRE interaction point. Thick connectors have been translated to instances of specific adapters. Finally for each component, a wrapper of its executable specification is generated as well as its IP-XACT description producing, after a compilation step, an executable simulator.

5. DESYREII SIMULATION BACKEND FOR DESYREML

As introduced in Section 4, the DESYREII framework is used to simulate and analyze the behavior of the DesyreML System. The framework has been integrated with the SysML import flow and a set of basic components (such as the Thick Connectors) to correctly support the import and the simulation of DesyreML compliant systems.

5.1 Backend Architecture for DesyreML

The DESYRE backend architecture for the DesyreML project integrates the semantics of SPEEDS with the ***Continuous Time*** (CT), ***Discrete Time*** (DT) and ***Discrete Event*** (DE) models of computation (MoC) supported by SystemC and SystemC-Ams. A set of possibly interconnected components defined over the same MoC will be called a cluster. The SPEEDS operational semantics permits to solve combinational loops between components that are part of the same SPEEDS cluster by fixed point semantics, when possible, as described in [14]. The SPEEDS semantics comprises two different MoCs: the ***Continuous Time Discrete Value*** (CTDV) and the ***Synchronous Reactive*** (SR). Fig. 4 shows the logical layered architecture of the DESYRE backend for DesyreML. Each model relies on the scheduler associated with its MoC. The different schedulers execute in an autonomous fashion. The DesyreML Synchronization layer has the purpose of synchronizing the different schedulers. Whenever an event occurs at the boundary between two components based on two different MoCs, the Synchronization layer ensures that the schedulers schedule the components in the correct order.

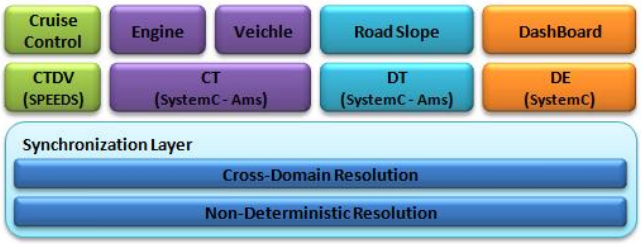

Figure 4 - DESYREII / DESYREML Layered Architecture

Future work will be done to support other MoCs such as Static and Dynamic *Data Flow*, *Asynchronous Systems* and *Stochastic Hybrid Systems*.

5.2 Implementation of operational semantics

To implement the operational semantics in DESYRE two main aspects have been considered: the scheduling of the different MoCs and the interconnections and adaptations of

signal exchanged between components based on different MoCs. A component consists of two main parts: the body and the wrapper. The body is the part of the component containing the specification of the model behavior and it shall be compliant with the component specific MoC. The wrapper is the part that allows the component to be handled by the DESYRE backend. Depending on the MoC, the component's body can assume different forms.

Continuous Time (CT) models consist of a composition of different continuous time object provided by the SystemC-Ams library, such as: *adders*, *subtracters*, *integrators*, *derivators*, *transfer function* and so on.

Discrete Time (DT) models are modeled as SystemC-Ams Time Discrete Function (TDF) modules. The *processing()* method of the module contains the body behavior definition while the *set_attribute()* method is used to set the component period and the I/O signals rate and delay.

Discrete Event (DE) models are modeled using SystemC methods sensitive to the events occurring on the input ports. The method contains the behavior of the component and is immediately executed whenever an event is notified.

Continuous Time Discrete Value (CTDV) models are the equivalent of the SPEEDS component containing only discrete flow ports. According to the SPEEDS MoC, those models are time triggered models where the time interval between two activations is not fixed and is decided by the component. The most significant part of the body is represented by the *step()* and the *commit()* functions. The *step()* function computes the component's outputs without updating its state. The *step()* function is used to resolve the fixed point semantics. The *commit()* function updates the component's state and communicates the output discrete flow values to the interconnected components. The calling order of those functions is decided by the SPEEDS scheduler and is defined in the SPEEDS hosted simulation protocol.

Synchronous Reactive Event (SRE) models are the equivalent of SPEEDS component containing both discrete and event flow ports. The body structure is similar to the CTDV one, with the *step()* and *commit()* functions, with the exception that event flows are communicated immediately to the interconnected components by the step() function.

The wrapper has the purpose to adapt the body, defined with different tools and languages, to the DESYRE framework. For each I/O port of the body, the wrapper contains a specific I/O port implementing an MoC-specific DESYREII interface. As mentioned before, modules with different MoCs exchange signals of different types in a different way. As a result, a semantic adaptation is required to interconnect two modules implementing two different MoCs. The adaptation is provided by special components called ***Thick Connectors*** that appear as simple parameterized connections in the SysML internal block diagram, as they are added by the model transformation flow.
For each pair of MoCs a specific *Thick Connector* has been implemented. Only one example of Thick Connector implementation will be reported in this article. Fig. 5 shows the internal structure of the *Continuous Time To Discrete Time Thick Connector*. The component consist of a first module that periodically samples the continuous time input signal and writes its value on a discrete time output signal of type double. The "Type Casting" module performs a casting from double to the data type of the DT port connected to the Thick Connector output. The sampling period is automatically set to the rate of the component connected to DT output port.

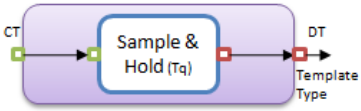

Figure 5 - CT to DT Thick Connector Structure

5.3 Cruise Control Simulation and Results

Let us now turn our attention to how the DESYREII framework is used to simulate and perform some analysis on the proposed use-case. As reported in Section 3.3, the DesyreML model transformation flow automatically generates a set of base components and XML files. The components represent the different modules that compose the cruise control system, while the XML files describe how those components are interconnected and configured. The component structure is generated according to the specifications reported in Section 4.1.
Continuous time modules have been simulated with an integration step of 2ms while, discrete time modules have been configured with a time period of 20ms.

The system has been simulated in steady state with an initial velocity of 130km/h and an initial throttle aperture of 0.324; the cruise control set point (desired velocity) is 130km/h. The road slope component is responsible for changing the status of the road slope during the simulation. Vehicle dynamic depends on the slope of the road (positive = road slanted upward, zero = flat road, negative = road slanted downward). Fig. 6 shows the road slope simulation profile. During the first 15 seconds, the road is flat. At 15 seconds the road starts slanting downward with a slope of -0.02. At 30 seconds the road takes up a positive slope of 0.06 and returns to a flat road after 15 seconds. Fig. 7 shows how the throttle aperture is controlled by the Cruise Control System in order to maintain the speed constant when the road slope changes. Fig. 8 reports the value of the speed during the simulation. It is possible to see that in coincidence of the points where the road slope changes, the velocity presents

an overshoot or an undershoot and then is stabilized again to the set point value.

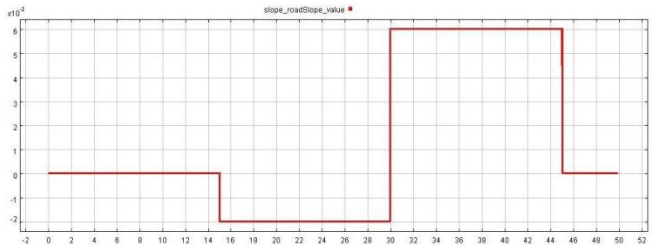

Figure 6 – Cruise Control System - Road Slope

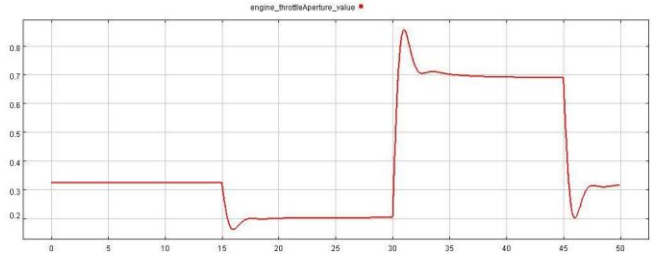

Figure 7 – Cruise Control System - Throttle Aperture

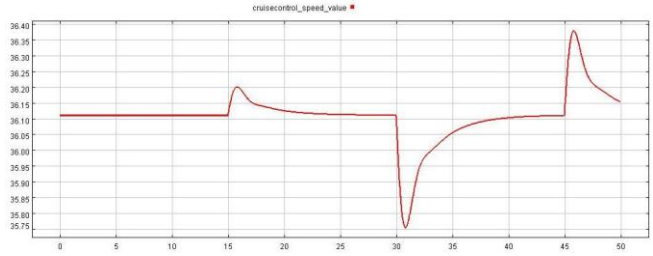

Figure 8 – Cruise Control System - Vehicle Speed

The simulation has been performed on an Intel® Core™2 Duo CPU P 9600 @ 2.66GHz with 4.00GB of RAM and Windows Vista 32-bit operative system. DESYREII required 1305ms to simulate 50seconds of the real system.

6. CONCLUSIONS

In this paper we presented the DesyreML integration language. The language allows for the integration of heterogeneous components supporting several models of computations (currently SR, DE, CT, DT, CTDV). The language syntax is based on the SysML language, using the profile mechanism to extend syntax and semantics. As an application of the language we described an analysis flow based on model transformation and the DESYRE analysis platform, integrating SystemC and SystemC-AMS, showing the capability of the language to describe an heterogeneous system through the Cruise Control use case.

As future work, we plan to extend the language and the analysis flow to allow the integration of other MoCs of interest, such as Asynchronous Systems (GALS), Markov Chain, Stochastic Hybrid Systems.

7. AKNOWLEDGMENT

This work was sponsored by DARPA under contract #FA8650-10-C-7074. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. The authors thank Prof. Alberto L. Sangiovanni-Vincentelli and the DARPA METAII project team for their valuable comments and discussions on the subject of this paper.

8. REFERENCES

[1] E. A. Lee and A. Sangiovanni-Vincentelli, “A framework for comparing models of computation”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1217–1229, December 1998.

[2] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli, "Composing heterogeneous reactive systems", *ACM Trans. Embed. Comput. Syst.* 7, 4, Article 43, August 2008.

[3] F. Balarin, A. Davare, M. D'Angelo, D.Densmore, T. Meyerowitz, R. Passerone, A. Pinto, A.Sangiovanni-Vincentelli, A.Simalatsar, Y. Watanabe, G. Yang, Q. Zhu, "*Platform-Based Design and Frameworks: Metropolis and Metro II*" in Model-Based Design for Embedded Systems, Boca Raton, Fl: CRC Press, Taylor and Francis Group, 2009, p. 259-289

[4] J. T. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "*Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems*", Int. Journal in Computer Simulation, 1994

[5] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, and C. Sofronis, *“A contract-based formalism for the specification of heterogeneous systems”* in FDL. IEEE, 2008, pp. 142–147.

[6] O. Ferrante, G. Codella, C. Sofronis, L. Mangeruca and A. Ferrari, *"Verify Contract-Based Designed Discrete Systems by Simulation"*, INCOSE EuSEC May 2010.

[7] SPEEDS project, http://www.speeds.eu.com/.

[8] Gerard, Sebastien, Selic, Bran, *"The UML MARTE Standardized Profile"*, Proceedings of the 17th IFAC World Congress, 2008.

[9] Herrera, F., Sánchez, P., and Villar, E. *"Heterogeneous system-level specification in SystemC"*. In Advances in Design and Specification Languages for SoCs, P. Boulet, ed. Springer

[10] J. Zhu, I. Sander, and A. Jantsch. HetMoC: Heterogeneous modeling in systemc. In *Proceedings of the Forum on Design Languages (FDL)*, Southhampton, UK, September 2010

[11] SysML language, http://www.omg.org/spec/SysML/1.2/

[12] ALES S.r.l., http://www.ales.eu.com/.

[13] QVT language, http://www.omg.org/spec/QVT/1.0.

[14] S. Edwards, E. Lee, *“The Semantics and Execution of a Synchronous Block-Diagram Language”* in Science of Computer Programming, 2003, vol. 48, no. 1, pp. 21–42.