EmbeddedMontiArc: Textual modeling alternative to Simulink

(Tool Demonstration)

Evgeny Kusmenko, Jean-Marc Ronck, Bernhard Rumpe, and Michael von Wenckstern
Software Engineering, RWTH Aachen University, Germany http://www.se-rwth.de

ABSTRACT

The development of complex cyber-physical systems relies heavily on elaborate domain specific languages, agile processes, and tools. A quasi-standard in multiple engineering domains is *Simulink*, a component & connector modeling tool by MathWorks. In this paper we present a versatile integrated development environment for EmbeddedMontiArc, a textual alternative for component & connector modeling. In order to deliver an agile and test-driven modeling experience, we integrate the complete tool-chain including code generators, compilers, as well as a variety of target runtime environments such as simulators behind a single user interface facilitating the creation, execution, and validation of cyber-physical system models.

KEYWORDS

C&C Modeling, Cyber-Physical Systems, Vehicle Simulator

1 INTRODUCTION

EmbeddedMontiArc is a Component & Connector (C&C) modeling language family developed with an emphasis on the needs of the embedded and cyber-physical systems domain [4]. It is inspired by the big players such as *LabView* and *Simulink*, but reveals a novel textual approach to component and connector modeling. The language family is composed of a multitude of modular textual languages using the MontiCore language workbench [3]. The main language elements include an architecture description language for subdividing a system into its components and sub-components, a matrix-based and strongly typed behavior specification language, a stream language for model-based testing, etc. One of the main strengths of EmbeddedMontiArc is its mathematical type system abstracting away from the technical realization. It integrates SI-units and unit-related compatibility checks as well as conversions directly into the language.

EmbeddedMontiArc provides an elaborate tool-chain transforming the original user-defined model to an executable binary. Its main steps include a parser, code generators for a series of target languages such as C++ and WebAssembly as well as compilation and linking [5]. As models are rarely developed for stand-alone usage as apps, but are rather meant to be used in heterogeneous target environments such as robot controllers, vehicles, simulators, and others, the resulting binaries often need to be deployed in a runtime environment for testing. In our tool demonstration we present *EmbeddedMontiArcStudio* (EMAS) - an integrated development environment (IDE) providing the aforementioned development tools out-of-the-box.

2 FEATURES

EMAS bundles all the features of *EmbeddedMontiArc* to a portable Windows 10 64-bit application or a Ubuntu 16.04 64-bit virtual machine enabling us to use *EmbeddedMontiArc* and its tool collection out-of-the-box. EMAS is publicly available under:

http://www.se-rwth.de/materials/embeddedmontiarc/

Similar to *Simulink* the main purpose of *EmbeddedMontiArc* is C&C modeling for embedded and/or cyber-physical systems. In contrast to *Simulink*, which is a graphical modeling language and stores multiple components (subsystems) in the same file, *EmbeddedMontiArc* is a textual modeling language family that, similar to Java classes, stores each component in its own file. Compared to graphical modeling, the textual modeling concept of *EmbeddedMontiArc* exhibits multiple advantages:

(1) All model information is directly available in files, and can be found and replaced by standard text programs. In graphical modeling tools such as *Simulink*, *Enterprise Architect*, or *PTC Integrity Modeler* this information is hidden behind a multitude of tabs or dialog boxes and often stored in proprietary binary formats. Integrated search speed for large models is slow.

(2) Text-based versioning tools like SVN and git featuring model differencing as well as merging and branching can be used out-of-the-box. Graphical tools allow one to export their graph structure as XML files, but reading an XML difference is hard since links between graphical elements are represented as links between unique identifiers.

(3) In large projects where multiple teams work together on the same model but on different files or even folders according to their responsibilities, *EmbeddedMontiArc* follows the separation of artifacts principle to enable modeling in the large. Having different files for different components enables a better integration into version control software since single components can be merged or reverted independently.

(4) Test driven development increases the code quality dramatically. *EmbeddedMontiArc* has an easy to use first level integration of unit tests for components whereas *Simulink* uses a graphical testing framework based on the signal builder. In *Simulink* executing multiple unit tests for one component is achieved by copying the signal builder and the component under test several times.

(5) In agile development, software is often updated to better fit customer needs. But updating large graphical models (e.g. inserting and reconnecting components) is very time consuming as the existing graphical layout needs to be rearranged manually. In *EmbeddedMontiArc* the graphical layout is generated automatically based on the textual files, serving a better understanding of the C&C architecture. This way, the modeler can focus on the main task by only adding, changing, or removing textual lines.

Figure 1 shows screenshots of the six main features of *EmbeddedMontiArc*: Ⓐ An interactive IDE with syntax highlighting, outline and parse error messages. The IDE support for all the languages of the *EmbeddedMontiArc* language family is automatically generated using the MontiCore framework.

Ⓑ A quality report for all *EmbeddedMontiArc* models. The tool shows all parse, resolve, and testing errors with its console output; it is a continuous integration/deployment CI/CD front-end for our modeling family. The tool can also be used stand-alone

Figure 1: Screenshots of Main Features of EmbeddedMontiArc: Ⓐ IDE, Ⓑ Quality Reports incl. Results of Unit Tests, Ⓒ Image Clustering, Ⓓ Generated Graphical C&C Layout of Textual EMA Models, Ⓔ 3d-Car Simulator Executing Autopilot Controller, Ⓕ Image Classifier based on CNN Models

(without EMAS) on a CI/CD server (e.g., travis-ci or gitlab runners) to perform model-based regression tests.

Ⓒ The simplest simulator of *EmbeddedMontiArc* is the *image clusterer*; it allows one to select pictures and converts them into a third order tensor object (the first two tensor degrees represent the height and the width of the image, respectively, while the third degree contains the color channels red, green, and blue) which is passed to the C&C model; finally, the simulator converts the resulting 2d matrix of the C&C model back to a gray-scale image.

Ⓓ The visualization generator produces a graphical representation of the textual *EmbeddedMontiArc* models as HTML and SVG files. A special feature of this generator is to produce graphical models with different abstraction levels, e.g. only showing the connected components but no ports up to drawing the complete model including all ports with their respective names and types. Since the layouts are computed automatically unlike in *Simulink* where only names or lines are hidden, the abstract graphical visualization is much smaller than the complete one and thus well suited to obtain an overview of large C&C models.

Ⓔ The car simulator [1, 2] with its 3d visualization and physics engine executes car controllers to test *EmbeddedMontiArc* models and their environment interactions. This feature is used for acceptance testing allowing users to judge the driving behavior of cars easily. Also the car simulator is a good motivation push for students to create *EmbeddedMontiArc* models as they can experience their models in the 3d visualization of the simulator.

Ⓕ The *image classifier* simulator works similarly to the *image clusterer*. It allows to drag (or to select your own) pictures into the analyze zone (white box) and then analyzes the content in the images and returns a predicted class (such as dog, cat, truck, and so on). The *image classifier* is a trained CNN (Convolutional Neural Network) component where the test images of the simulator are of course different from the training data.

All three simulators Ⓒ, Ⓔ, and Ⓕ have in common that a C++ compiler toolchain translates the textual EMA models to native code (dynamically linked libraries or standalone executables) before the respective simulator can execute the model. A feature of the toolchain is that it includes the highly optimized Armadillo mathematics framework guaranteeing an efficient model execution of computationally intensive software. This enables the simulation of different cars at the same time and on the same laptop, the execution of expensive algebraic algorithms in our image clustering example, as well as the classification of images at nearly real-time on standard customer hardware. Fast simulation results are a key to keep students motivated to test their models and thus improve them continually. For instance, our vehicle visualization works at 30 fps, image classification is executed without a noticeable delay.

3 EMBEDDEDMONTIARC MODEL EXAMPLES

Figure 2 presents code snippets used in three of our executable model examples: the **Autopilot** Ⓔ model snippet ① belongs to the `Autopilot` model. This component checks whether the car is still in the track. This component has two input ports (ll. 2-3) for the car position relatively to the track accepting values between -200 meters and +200 meters, and one Boolean output port (l. 4) returning `true` when the car is outside the track boundaries. The implementation part of the atomic component (ll. 5-8) contains code defining how sensor inputs are mapped onto a Boolean output.

Testing: ② is a blackbox unit test based on the stream semantics. The concept is very similar to JUnit. A stream test operates

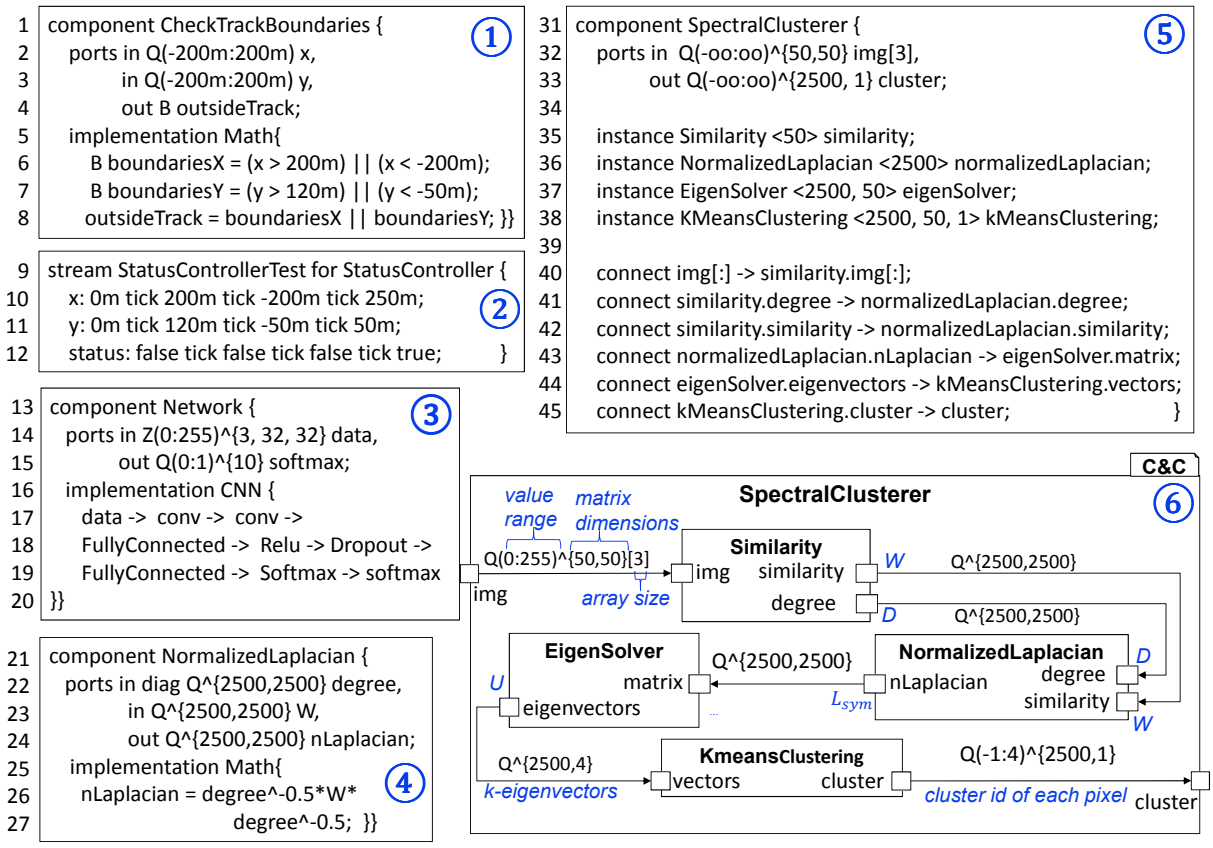

Figure 2: Concrete syntax examples of *EmbeddedMontiArc* language family: ① simple component checking if car is inside the track; ② unit test of ①; ③ component using CNN; ④ component executing matrix operations; ⑤ complex component being decomposed into subcomponents. Picture at the right bottom is visual representation of ⑤.

on exactly one component specified after the `for` key word (l. 9). Of course, multiple stream tests can exist for each component. The first lines (ll. 10-11) provide the values for the input ports of the components while the last line (l. 12) specifies the expected values of the output port. The stream language also supports underspecification, e.g. by defining ranges for output values. Iff the component is executed and the calculated output values satisfy the expected values of the stream, then the test will be considered as *pass*.

Deep Learning Classifier Ⓕ: In ③ a component using an alternative implemention language, namely *CNNArch* is presented. It supports a compact description of layered artificial neural networks which become more and more ubiquitous in intelligent software systems. The CNN model describes the layers of the network as well as their interconnections. A generator then generates the target network architecture, trains the network on a given labeled data set and encapsulates the software in a standard *EmbeddedMontiArc* component. Using a simulator similar to the clusterer users can check the quality of the classification on their own images.

SpectralCluster Ⓒ: the model snippets ④ and ⑤ belong to the spectral clusterer model, which can be used to detect objects in a picture. Since *EmbeddedMontiArc* is a textual modeling language the developer writes down the model shown in ⑤, which is semantically equivalent to the C&C model displayed in ⑥. The `SpectralCluster` component is decomposed into four subcomponents (ll. 35-38). The information between these four subcomponents is exchanged via unidirectional connectors (ll. 40-45). Model ④ shows the implementation of the subcomponent `normalizedLaplacian` (l. 36) having the type `NormalizedLaplacian`. Lines 26 and 27 show that the implementation part of *EmbeddedMontiArc* models support normal matrix-vector expressions similar to MATLAB code.

REFERENCES

[1] Christian Frohn, Petyo Ilov, Stefan Kriebel, Evgeny Kusmenko, Bernhard Rumpe, and Alexander Ryndin. 2018. Distributed Simulation of Cooperatively Interacting Vehicles. In *21st IEEE International Conference on Intelligent Transportation Systems (ITSC'18)*.

[2] Filippo Grazioli, Evgeny Kusmenko, Alexander Roth, Bernhard Rumpe, and Michael von Wenckstern. 2018. Simulation Framework for Executing Component and Connector Models of Self-Driving Vehicles. In *Proceedings of MODELS 2017. Workshop EXE*.

[3] Katrin Hölldobler and Bernhard Rumpe. 2017. *MontiCore 5 Language Workbench Edition 2017*. Shaker Verlag.

[4] Evgeny Kusmenko, Alexander Roth, Bernhard Rumpe, and Michael von Wenckstern. 2017. Modeling Architectures of Cyber-Physical Systems. In *ECMFA*.

[5] Evgeny Kusmenko, Bernhard Rumpe, Sascha Schneiders, and Michael von Wenckstern. 2018. Highly-Optimizing and Multi-Target Compiler for Embedded System Models: C++ Compiler Toolchain for the Component and Connector Language EmbeddedMontiArc. In *MODELS*.