# A Scade Suite to Modelica Interface

Daniel Schlabe, DLR Institute of Robotics and Mechatronics - Daniel.Schlabe@dlr.de
Tobias Knostmann, Esterel Technologies GmbH - Tobias.Knostmann@esterel-technologies.com
Tilman Bünte, DLR Institute of Robotics and Mechatronics - Tilman.Buente@dlr.de

## Abstract

This article presents implementation and utilization details of the currently developed interface from Scade Suite to Modelica. By a few clicks one can generate a Modelica block from Scade Suite models that can be directly used and simulated in Modelica. This block calls an external function periodically, where the C-code generated by Scade Suite is invoked.
The main purpose of the interface is to test the generated C-code within a simulated environment which is also known as Software in the Loop (SIL).

*Keywords: Scade Suite; Modelica interface; C-code integration; Software In the Loop*

## 1 Introduction

### 1.1 Scade Suite Description

The acronym Scade stands for Safety-Critical Application Design Environment. The term describes a Suite of model-based software development and verification tools as well as the modelling language itself. This language is formally defined and proven to be fully deterministic, hence it allows certified and qualified code generation for safety-critical systems. The textual base of the language is an extension [3] of the synchronous dataflow language Lustre [10]. The textual language is by default hidden behind a design environment for graphical models featuring deterministic state machines, dataflow block diagrams and decision diagrams.
Most importantly, this formal technology allows to create a seamless process to leverage a large degree of automation:

- model editing
- verification by means of simulation and formal methods
- traceability
- documentation generation
- code generation

It also enables circumvention of cumbersome activities demanded by today's critical software standards such as DO-178B, EN50128 or IEC61508 (see [5] and [6]).
The qualification and certification of the Scade KCG code generator demands that the produced code is not altered in any way. Therefore, in order to adapt the code regarding specific calling conventions of e.g. Modelica functions, the developer needs to keep the KCG code unchanged and add C-code to meet the interface requirements.
To automate the generation of this glue code, Scade offers a Tcl-based [12] scripting environment that runs in parallel to the code generation process. It offers access via a specific application programming interface (API) to the model structure information and the translation patterns of Scade-language based objects to C constructs. This method is used to provide adaptors to various real-time operating systems and other code-wrapping targets, all of which can be adapted to the users needs. We will describe how we use this means to create our Modelica adaptor.

### 1.2 Scope of the interface

Thinking of a Scade Modelica interface one can imagine two possible interface "directions". The first one is to integrate Code generated by Scade Suite in Modelica. The second one is to generate a Scade Suite model out of a Modelica model. The herein developed interface deals with the first interface direction. Certainly, the second direction is very interesting since the generated Scade code would be certified. But this will still need a lot of investigations and work. Furthermore it would be restricted to a subset of Modelica and not the whole language. See chapter 4 for an outlook of this point.
By using the herein developed Scade-Modelica interface it is rather possible to test the generated C-code at a very early stage of the design process

by means of simulated environments, also known as Software in the Loop (SIL). This will reduce development time since it is not always necessary to implement the code on the target system. Furthermore it is possible to specify requirements beforehand by means of Modelica model environments that contain simple placeholders for the functions to be developed. This will reduce iteration loops with the client who specified the requirements and enables therefore a fast software development.
The integration of any C-code into Modelica could be done manually indeed. However, this requires creating and modifying interface files like C-code or Modelica code every time the model is changed, which is quite inconvenient. Using the developed automatic interface one can easily integrate new C-code into Modelica and test it in the simulated environment.

### 1.3 Application Areas

Typical application areas will be the aerospace and automotive industry or any other field where a safety critical function should be integrated into a complex system. Any developed function or controller can be tested in a simulated environment or object using the interface. A Modelica block can easily be generated as long as the Scade model fulfils the restrictions described in section 2.1. For instance, energy management for more/all electric aircraft or hybrid/electric vehicles is a topic of rising interest that becomes more and more complex, where interactions with the system model as well as with other management functions or controllers can often not be neglected. Therefore it is useful to be able to simulate the management function together with the aircraft or vehicle model.

## 2 Implementation

### 2.1 Basic principle and restrictions

The basic principle of the developed interface is to integrate C-code into Modelica through external functions. Therefore, an adaptor for code integration to Modelica has been developed for Scade Suite. So the KCG-code can be used without any changes. The adaptor generates some additional interface files and a Modelica package. This folder contains:

- Unchanged KCG-code
- Additional C-code for the interface function and for Scade state vector (see next sections for more details)
- An image
- Modelica package containing one package.mo file

The additional code, the folders and the .mo files are generated via Tcl scripts. These scripts can be easily executed during code generation in Scade Suite.
For the current version of the interface the following restriction applies: only Integer, Boolean and Real are allowed as input and output signal types. That means that for each input a separate connector will be displayed in Modelica. On the one hand, this eases readability. But on the other hand, with more complex models having many inputs and outputs, the Scade block will be very large. A vector of Boolean, Integer and Real could technically be possible, but this would seriously affect readability since the variable would be identified with an index instead of its original name. For this reason, this method was not implemented in the current version. Also, records are not supported yet. In fact, Dymola doesn't support records in external C-code correctly.
Provided that the input and output signals are of the supported types as described previously, there is no further restriction regarding complexity or internal states of the Scade Model. The interface will also work for large numbers of inputs and outputs, though the resulting block illustrated in Modelica will be very large as well.

### 2.2 Implementation Details

The implementation details will be illustrated by way of an example. Figure 1 displays the Scade model of a

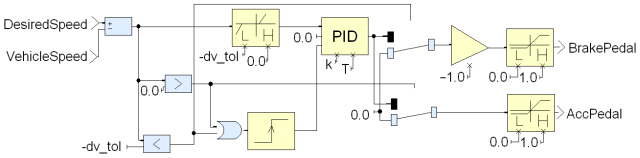


Figure 1: Speed Controller in Scade Suite

vehicle speed controller that outputs brake and acceleration pedal positions using the current speed of the vehicle and the desired speed as an input. The Modelica adaptor for Scade Suite will generate an additional tab. As shown in figure 2 just three items are required:

- The target directory,
- Modelica project name = Modelica package name,

- The Modelica version to be used.

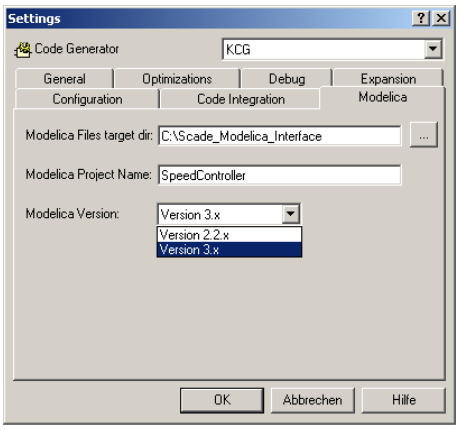


Figure 2: Modelica Tab in Scade

The interface is tested for Modelica versions 2.2.2, 3.0 and 3.1. There were some changes in the annotation syntax between Modelica 2.2.2 and 3.0, which the interface takes into account. So if a wrong Modelica version is chosen, important graphic elements like the connectors will not be displayed.
The respective C-files and a Modelica Package are now generated containing the following elements:

- block ScadeBlock
- class ScadeStateVector
- function ScadeStep

The basic element is the function ScadeStep, where the C-code is integrated via external functions:

```
function ScadeStep
  input Real vehicleSpeed ;
  input Real desiredSpeed ;
  input ScadeStateVector ssv;
  output Real brakePedal ;
  output Real accPedal ;
  external "C" ScadeStep(vehicleSpeed,
desiredSpeed, ssv, brakePedal, accPedal);
  annotation(Include="
    #include <../SpeedControl.c>
    #include <../PID_BaseClass.c>
    ...
",
 uses(Modelica(version="3.0")));
end ScadeStep;
```

The C-code is included using absolute paths. So the working directory doesn't need to be the package directory. This implies that one has to modify the absolute paths here manually if the package is moved or copied elsewhere.
The inputs and outputs of the Scade model are mapped one-by-one to the Modelica function. If the Scade model has any dynamic states one extra input is needed for the function: the so called Scade state vector. Since functions in Modelica are not allowed to have any internal state, the previous state will be an input for each call of the function. Going back to the speed control example, the PID controller has some internal states. The corresponding Scade state vector is specified in Modelica as follows:

```
protected class ScadeStateVector
"External Scade States"
  extends ExternalObject;
  function constructor
    output ScadeStateVector ssv;
    external "C" ssv =
initScadeStateVector();
  end constructor;
  function destructor "Release storage of
ScadeStateVector"
    input ScadeStateVector ssv;
    external "C" freeScadeStateVector(ssv);
  end destructor;
end ScadeStateVector;
```

Using external C-code generated by the Modelica adaptor, memory is allocated by the constructor and freed by the destructor. The ScadeStep function and the ScadeStateVector class are declared protected since the user shouldn't use them directly.
The user interface in Modelica is represented by a Scade block (see figure 3) that can be used per drag and drop. It just needs the sample period and a starting time as parameters. In this Block the function


Figure 3: Scade Block in Modelica

ScadeStep will be called periodically as specified. Furthermore the Scade state vector will be initialized herein:

```
block ScadeBlock
"Scade Suite Block containing standard
interfaces generated by Scade"
```

```
protected
ScadeStateVector sSVector=
ScadeStateVector();   // call init
annotation (...)
public
parameter SI.Time samplePeriod(...)
  "Sample period of component";
parameter SI.Time startTime=0 "First
sample time instant";
protected
output Boolean sampleTrigger "True, if
sample time instant";
output Boolean firstTrigger "Rising edge
signals first sample instant";
public
// ----------- Inputs ------------
  Modelica.Blocks.Interfaces.RealInput
vehicleSpeed annotation (...);
  Modelica.Blocks.Interfaces.RealInput
desiredSpeed annotation (...);
// ----------- Outputs ------------
  Modelica.Blocks.Interfaces.RealOutput
brakePedal annotation (...);
  Modelica.Blocks.Interfaces.RealOutput
accPedal annotation (...);
equation
  ...
  when {sampleTrigger, initial()} then
  (brakePedal ,accPedal)=ScadeStep(
vehicleSpeed, desiredSpeed, sSVector);
  end when;
end ScadeBlock;
```

The display size of the block as well as the connector locations are adapted to the specified number of inputs and outputs in the annotations.
The next chapter will show the functionality of the Scade block in a realistic model.

## 3 Automotive Example

As a realistic application example we chose the speed control of a multibody vehicle model (see figure 4) using the speed controller previously implemented in Scade as already presented with figure 1 in section 2.2. The total model is set up in the framework of the VehicleInterfaces library and uses some of the component classes provided there [4]. As vehicle dynamics model (*chassis* in figure 4) we used a multibody vehicle model from the DLR VehicleControls library [9]. Passengers are represented by multibody models as well.

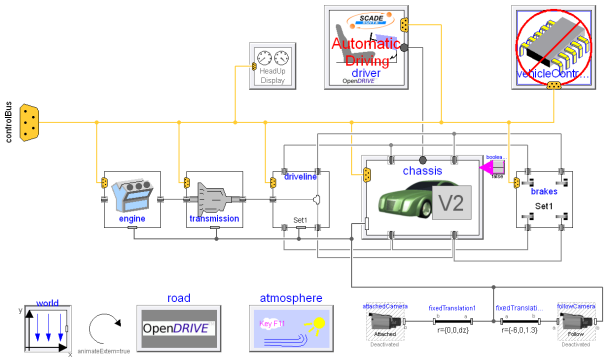


Figure 4: Total Model of the Vehicle in Dymola

The road definition complies with the OpenDRIVE standard [11] while the used commercial OpenDRIVE database plus visualisation was purchased from Vires Simulationstechnologie GmbH. During the simulation the vehicle drives along the OpenDRIVE road. Automatic steering control is used for lane keeping. The focus of our simulation, however, is on the automatic speed control implemented (see figure 1 ) using the Scade interface as shown in figure 5. The algorithm

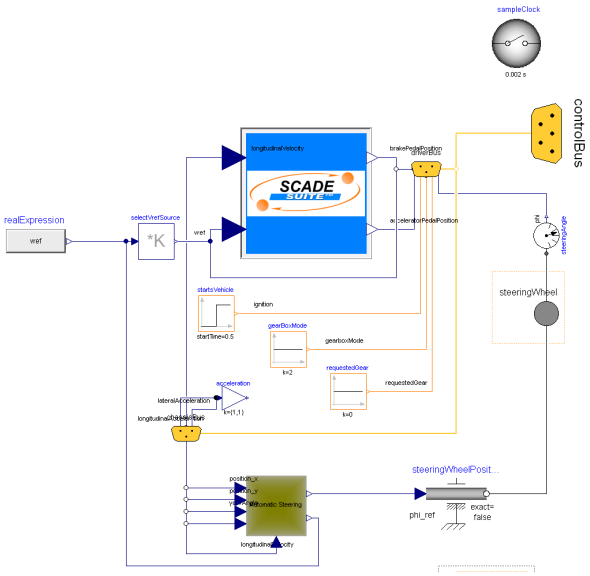


Figure 5: Scade Block integrated into Driver Model

used to calculate adequate gas pedal and brake pedal positions was adapted from a class out of the DLR PowerTrain library [7]. The desired vehicle speed used in the controller comes from an algorithm which computes a *reference speed profile* for the upcoming road section. Details can be found in [2]. The simulation results from the automatically driving vehicle using the imported Scade speed controller were well matching our expectations. The assessment was supported online by animation using components and the external

viewer *SimVis* provided with the DLR Visualization library.

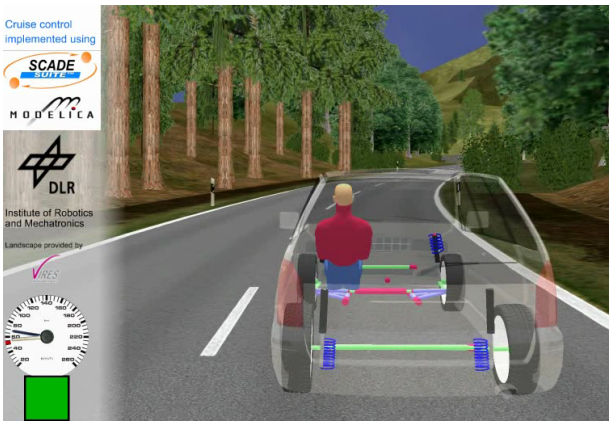


Figure 6: Visualisation in SimVis

## 4 Conclusion and way forward

A first version of an interface from Scade Suite to Modelica has been presented in this paper. The interface will be adapted to future versions of Modelica and Scade Suite if needed. One planned enhancement is the removal of absolute paths for including C-code. This can be replaced by URIs introduced in Modelica version 3.1. It is also possible to use the Functional Mock-Up Interface (FMI) instead of external functions for future versions.
Another very interesting thought is to generate ScadeModels directly out of Modelica. This would enable the automatic generation of certified code from a Modelica model. Certainly it can be expected that some restrictions will apply to the Modelica model. This opposite interface direction can be a challenging task for future investigations.

## 5 Acknowledgements

This work has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) for the Clean Sky Joint Technology Initiative under grant agreement n° CSJU-GAN-SGO-2008-001 [8].

## References

[1] Bellmann, Tobias (2009) Interactive Simulations and advanced Visualization with Modelica. In: Proceedings of the 7th International Modelica Conference. Linköping University Electronic Press. Modelica Conference, 20.-22. Sept. 2009, Como, Italien. ISBN 978-91-7393-513-5. ISSN 1650-3740

[2] Bünte, Tilman; Chrisofakis, Emanuel (2011) A Driver Model for Virtual Drivetrain Endurance Testing. In: Proceedings of the 8th International Modelica Conference. Linköping University Electronic Press. Modelica Conference, 20.-22. March 2011, Dresden, Germany.

[3] Colaço, JeanLouis; Pagano, Bruno; Pouzet, Marc (2005) A Conservative Extension of Synchronous Dataflow with State Machines. In: EMSOFT'05 Sept. 9-22 2005, Jersey City, New Jersey, USA.

[4] Dempsey, Mike. An introduction to the VehicleInterfaces package. Tutorial at Modelica conference 2006, Vienna, 2006.

[5] Fornari, Xavier. Understanding How Scade Suite KCG Generates Safe C Code. 2010. White Paper of Esterel Technologies. [online]: http://www.esterel-technologies.com/technology/WhitePapers/

[6] Pagano, Bruno; Andrieu, Olivier; Moniot, Thomas; Canou, Benjamin; Chailloux, Emmanuel; Wang, Philippe; Manoury, Pascal; Colaço, Jean-Louis. Experience Report: Using Objective Caml to develop safety-critical embedded tools in a certification framework. In: Proceedings of the 14th ACM SIGPLAN international Conference on Functional Programming. Edinburgh, Scotland, August 31 - September 02, 2009

[7] Tobolar, Jakub; Otter, Martin; Bünte, Tilman. Modelling of Vehicle Powertrains with the Modelica PowerTrain Library. In: Systemanalyse in der Kfz-Antriebstechnik IV, Seiten 204-216. Dynamisches Gesamtsystemverhalten von Fahrzeugantrieben, Augsburg, 2007.

[8] Clean Sky project homepage [online]: http://www.cleansky.eu

[9] EUROSYSLIB Project Profile 2007 [online]: http://www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf

[10] The synchronous dataflow programming language LUSTRE (1991) [online]:

http://citeseer.ist.psu.edu/viewdoc/
summary?doi=10.1.1.34.5059

[11] OpenDRIVE project [online]
http://www.opendrive.org

[12] Tool command language (Tcl). [online]:
www.tcl.tk or http://citeseer.ist.psu.edu/viewdoc/
summary?doi=10.1.1.38.8230