Gene-Auto: an Automatic Code Generator for a safe subset of Simulink/Stateflow and Scicos

A Toom, T Naks, Marc Pantel, M Gandriau, I Wati

Gene-Auto: an Automatic Code Generator for a safe subset of Simulink/Stateflow and Scicos

A. Toom[1,4], T. Naks[1,4], M. Pantel[2], M. Gandriau[2], I. Wati[3]

1: IB Krates OÜ, Akadeemia tee 19, 12618 Tallinn, Estonia
2: IRIT-ENSEEIHT, University of Toulouse, 2, rue Charles Camichel, 31071 Toulouse Cedex, France
3: CRIL Technologies, Alyotech France, 104 Boulevard Auguste Blanqui, 75013 Paris, France
4: Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia

Abstract: Model Driven Engineering (MDE) and Automatic Code Generation (ACG) have been very successful in the systems engineering domain. An important aspect, however, is that the required lifetime of critical embedded systems can be very long. For instance, avionic software must be maintainable for up to 80 years. Secondly, the reliability of high criticality software must be qualified according to high industry standards (e.g. DO-178 in the avionics). The Gene-Auto project addresses these issues by developing an open-source toolset for code generation from high level modelling languages like Simulink/Stateflow and Scicos to executable code for real-time embedded systems. It is the goal of the project to prepare the ACG for full qualification according to the requirements of safety critical industries. However, since the traditional test-based verification methodologies are very costly, Gene-Auto develops and aims to qualify some parts of the toolset using novel formal technologies.

Keywords: Model Driven Engineering (MDE), Automatic Code Generation (ACG), open-source, qualified toolset, Simulink/Stateflow, Scicos.

1. Introduction

Model driven engineering emphasizes the use of models to raise the level of abstraction in the design of complex systems and model transformations that eventually lead to executable code [1]. The assumption is, that application specific constructs and solutions can be more easily described using a model than a programming language. Higher level of abstraction enables to verify/validate solution before adding implementation details. Model transformations are intended to ensure that solution described in an abstract model is carried to the implementation correctly and without losses.

A specific kind of model transformation that produces executable program code is referred to as "code generation" (or "automatic code generation" to emphasise that the process involves little or no intervention from the programmer). The goal of the approach is to increase the level of software quality by suppressing routine phases of development (coding, unit testing) and also reduces the cost of software development by shortening the development process. The latter may not apply in all cases, as the input models are required to be much more detailed and consistent than the models that are created only for communication purposes and are interpreted by human programmers. However, in the safety critical domain, the increased level of correctness is still an advantage of the model driven development, even if some drawbacks in the development time might occur. Second target for cutting the length of development process is testing. Provided that the solution is verified/validated on the model and the transformation from model to code is proven to be correct, some phases of testing can be potentially reduced or completely removed. Additionally, models allow to express separately different concerns – e.g. functionality, performance, safety, etc. All of these will be then merged in the source code and the binary executable.

The design models used in the systems engineering, that includes also safety critical domains like aerospace, automotive, etc, are often in the form of functional data-flow diagrams and finite automata. One of the most widely used tools in this domain are Simulink [2] and Stateflow [3], belonging to the Matlab family. An open source tool similar to Simulink is Scicos [4], part of the Scilab toolset. However, often such tools have been designed for a much wider audience and allow usages, which do not always fit the quality requirements of safety critical systems. Examples of formalisms that were specifically developed for the safety critical domain are the languages belonging to the synchronous language family such as Esterel [5], Lustre [6] and Signal [7] and Lucid-Synchrone [8], which led to the industrial products Esterel Studio [9], SCADE Suite [5] and RT-Builder [11].

This paper describes the aims and current results of a project on an Automatic Code Generator for the safety critical domain – Gene-Auto. The purpose of the Gene-Auto project is, on one hand, to define a modelling language which is a safe subset of Simulink/Stateflow and Scicos and fits the needs of its industrial partners, and on the other hand to specify and implement an ACG taking as input this modelling language and producing as output a safe subset of the C language, corresponding to the usual coding rules for real-time embedded safety

critical systems. As the ACG will produce the source code for the system without any humans in the loop, it is currently required by some certification authorities to have the same safety level as the system it is used for. Current testing technologies make reaching this level of safety in an ACG very laborious. Gene-Auto will therefore rely as much as possible on formal technologies to ensure the quality of the toolset.

The motivation to start another project for creating code generation in a domain, where several industrial-grade tools exist already, is manifold. The first and main aspect is the required longevity of support. For instance, in the aerospace domain the required lifetime of control software (and thus also the lifetime of all tools required to support this software) is up to 80 years. It is difficult to expect from a single organisation to guarantee support for such a long period. One possible solution is the open-source approach. Having access to the source code of a software product will allow to maintain this product after the end of support from its original producer. The second requirement is related to tool qualification. As already mentioned above, when the software tool will replace some of the activities performed by human programmers, the tool must conform to the same safety level than the developed software. This means automatically that the tool development procedure should follow specific rules and detailed information about the development process must be available for the certification activities. It takes considerable effort to follow all necessary procedures for qualification and usually tool developers (with some exceptions) do not find it cost effective or even feasible. The Gene-Auto consortium has a goal to follow the required development process and prepare all necessary records for the future qualification of the tool in automotive and aerospace domains.

This contribution will present the Gene-Auto project and its various parts. It will mainly focus on the proposed input language and on the practical and theoretical design constraints that are used in order to help the users in writing correct and easy to understand and maintain models, and on the other hand to provide a clean semantics, well suited to formal verification and validation technologies both for the models and the ACG.

2. The languages

2.1 Requirements and approach

The main purpose of the Gene-Auto toolset is to implement a code generator transforming a set of high-level graphical modelling languages to a textual programming language that will be used for developing safety critical real-time embedded systems. Initially, the Simulink, Stateflow and Scicos languages have been chosen as Gene-Auto input formalisms and C as the target language.

There are two main groups of requirements specified for the Gene-Auto toolset:

- User requirements, which define the subsets and semantics of both the input and output languages of the code generator and its qualification constraints.
- Tool requirements, which define the relation between the input and output languages that must be satisfied by the developed tools and their qualification plan.

Additionally to the input and output languages, two intermediate languages have been defined: one related to the input modelling language and the other related to the output language. The purpose of the first one is to generalise the common concepts found in the tools used for developing safety critical embedded systems and provide them with clean rigorous semantics. This language is called Gene-Auto modelling language. The other one is called Gene-Auto code modelling language and it is semantically close to the intended target languages of the code generator. The definition of these intermediate languages includes abstract syntax and semantics, but also modelling and programming rules that are meant to ensure that the models are well formed and the output code conforms to the coding requirements of end-users.

2.2 The Gene-Auto modelling language

2.2.1 Requirements and objectives

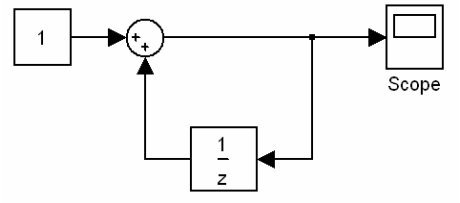
Most of the modelling languages widely used for designing safety critical embedded systems, have been designed for much broader purposes (for example UML2 or Simulink/Stateflow). Therefore, they provide great expressiveness and rely on complex semantics, which is usually not formally defined and changes from version to version of the tools or standards (UML2 even provides standard semantics variation points). Similarly to previous works (e.g. [12]) Gene-Auto defines a restricted subset of the input languages that has a simple, well-defined semantics. Such a subset is “safe”, because it avoids complex error-prone constructs and allows for simple verification of intended properties, e.g. termination, context independency, while still preserving the essential features of the language. The developed language must not only suit the needs of the various partners of the project but also be the basis of an open standard promoted by Gene-Auto. It serves as a basis for independent model verification tasks (including conformance to the chosen subset of the input formalism), as well as further refinement in the course of code generation (e.g. type or clock inference). However, it must be pointed out that it is not the purpose of the Gene-

Auto project to provide means for graphical editing and simulation. In fact, the effort for defining a common modelling language is similar and related to the larger open source and open architecture project for critical embedded systems TOPCASED [13].

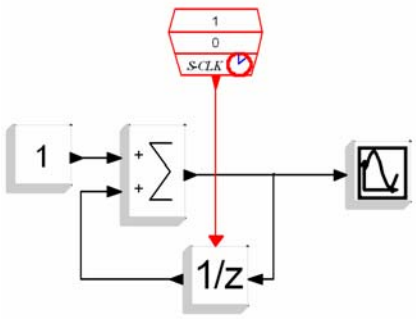
The Gene-Auto modelling language's semantics is based on the Kahn synchronous network model of computation [14]. Motivated by the commonly used graphical notations, this language defines four kinds of hierarchical diagrams: functional, automata, truth table and decision network, plus an action language. Following is a brief informal description of these languages along with some simple examples from different modelling tools.

2.2.2 The functional diagram

This diagram is the classical data-flow widely used in the systems engineering to express command and control systems. Signals carrying values are interconnecting blocks through their input and output ports. Each block can either be a basic (atomic) block or a subsystem. Each block can have a clock, an activation port and an execution rank. The clock is used to relate all the blocks, which will be synchronised implicitly. Two blocks with different clocks can only be connected through explicit rate adapter blocks. The activation port is used to decide if a block must be executed when its clock is ticked. The execution rank is an integer which must be unique in a given diagram and allow to decide the order in which the blocks with the same clock in a diagram are executed in a given clock tick. The basic blocks provide a combinatorial or sequential function handled by the code generator or provided by libraries following the Gene-Auto interface. Subsystem blocks can contain any diagram whose clocks are sub-clocks of the block ones.



(a)



(b)

Figure 1: Functional diagram models in Simulink (a) and Scicos (b)

2.2.3 The automata diagram

This notation is derived from the Harel's StateCharts [15] and is widely used in many modelling languages (UML2, Statemate, Stateflow, etc.). Gene-Auto proposes both Moore (actions in states) and Mealy (actions on transitions) models as well as parallel automata relying on the synchronous semantics. The potentially non-terminating "run-to-completion" semantics together with the intricate "early return" mechanism in Stateflow ([3], [12]) are avoided.

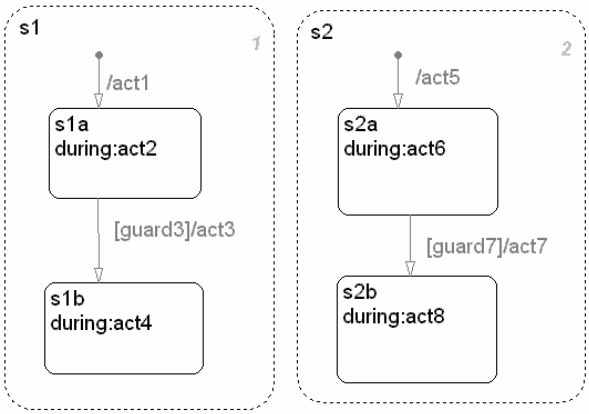


Figure 2: A chart with parallel automata in Stateflow

2.2.4 The truth table diagram

This diagram is the classical boolean algebra truth table. A cell is selected depending on the values of boolean expressions related to the values of input variables. Cells contain actions, which are executed when a particular cell is selected.

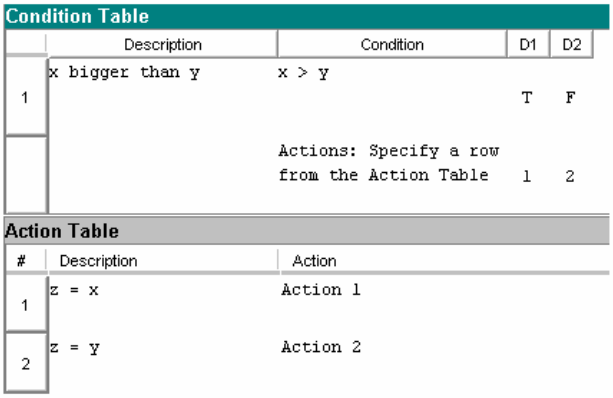


Figure 3: A truth table in Stateflow

2.2.5 The decision network diagram

A so-called decision network is a diagram that is a combination of junctions and transitions. It is derived

from the junction notation used in UML2 Statecharts and Stateflow diagrams. A junction is a kind of a step in a sequential execution. Actions are associated to the transitions between junctions. A transition is taken and the corresponding actions are executed, when the transition's guard evaluates to true.

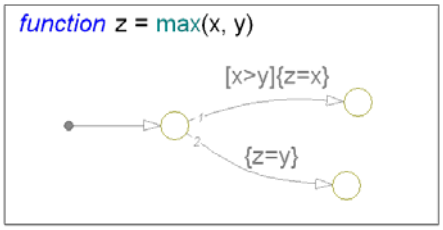


Figure 4: A graphical function in Stateflow

2.2.6 The action language

The purpose of the action language is to achieve some noticeable effect with the computation. Actions are associated to transitions, states of an automaton and cells of a truth table. The action language is a small imperative language that is a subset of the output language of the code generator.

2.3 The Gene-Auto code modelling language

The initial output of Gene-Auto code generator is an abstract language that captures the required subsets of the intended target languages. Currently only imperative languages like: C, C++, ADA and Java are considered. This language provides the minimum set of concepts required to generate efficient code: expressions, function/macro definitions, function/macro calls, read and write to a variable, definition of global/local variables, sequence, conditional statements, error handling, etc. The abstract output language is also a basis for further optimisation and verification tasks that can/will be performed before converting to specific concrete code. During the first phase of the Gene-Auto project only C language output will be generated.

3. Toolset architecture

The Gene-Auto toolset is an open-source customisable framework for generating software code from models of safety critical embedded systems. The scope of the Gene-Auto project is currently initiated for the generation of C-code from safe subsets of Simulink, Stateflow and SciCos modelling languages. However, the architecture supports adding both input and output languages. Another important goal is the validation and verification of the code generation process and support for model verification. For the end-user the Gene-Auto process is one-directional and all information needed for the code generation process is specified either in the source model or as code generator parameters. However, the toolset architecture supports exporting models between transformation steps for validation and verification and/or for optional transformation steps using add-on tools.

To allow for the maximum flexibility of tool development and qualification, the toolset architecture is modular and each module is an isolated functional component. Smaller components make specifying and verifying the properties of each component easier and make the whole toolset less sensitive of the implementation details of individual components. The requirement of pure functional nature (the behaviour of each component depends only on its input) makes it possible to verify and validate each component in isolation and combine the components later without any side-effects. In addition to reducing the verification effort, this also gives the possibility to use different verification and validation methods in different components. Gene-Auto intends to qualify selected parts of the code generator using formal methodologies, as a replacement for testing used in classical methods. As this is a challenging task, it will not be applied to the whole tool-chain at once, but rather, in the first Gene-Auto production version there will be parts that are qualified using traditional test-based techniques together with parts developed and verified using formal techniques. As the architecture is fully modular, each component can be replaced by a different one, yielding in time a code generator that has been fully built and verified using formal techniques.

There are several classes of general functionalities proposed in the tool-set architecture:

- Model importers – components that enable to import a source model from its native format and convert it to the Gene-Auto modelling language.
- Model transformers – components that do a specific transformation step on either a model in the Gene-Auto modelling language or a model in the Gene-Auto code modelling language. The output of such a transformation is again a model in either one of these languages, until the model is ready for printing out to concrete programming language code.
- Model verifiers – components that enable to validate the correctness of the model and correctness of the individual transformation steps. Validation of the model itself is out of the scope of the Gene-Auto tool-set, except to check that the model conforms to the required modelling constraints.
- Model serialisers – components that are coupled to both of the Gene-Auto model formats: the modelling language format and the code

modelling language format, and manage the transformation between the model and a file. Model serialisers are a compulsory feature of the implementation rather than tools on their own. They guarantee that the model can be losslessly stored and retrieved between the individual transformation steps.

In addition, a set of external modelling assistants and verification/validation tools are envisaged that are outside of Gene-Auto scope but are necessary for verification and validation of the input model. External tools can interface with the Gene-Auto toolset due to the open modelling languages.

4. Formal verification

Gene-Auto relies on proof assistant technologies for the validation and verification of the toolset. Proof assistants allow to build both: specifications and proof of properties about specifications that are correct by construction. The specifications are very similar to functional programming languages. Data are expressed as term algebra. Properties are expressed by pattern matching and induction/co-induction on the data structure.

Proof assistants are used in the Gene-Auto project to:

- Write formal specifications of:
  - Abstract syntaxes, design rules and execution semantics (user requirements) of the input and output languages
  - The relationship between the input and output languages (tool requirements)
  - Coherence properties of the different parts of the specification
  - Completeness properties of the different parts of the specification
  - Code generation tools
  - Model and code verification tools
  - Soundness properties of the code generation
  - Soundness properties of the model and code verification tools
- Design a proof that:
  - The specification is coherent, complete and sound
  - The tools both for generation and verification are sounds
- Generate automatically the implementation of the tools from their specification and soundness proof
- Generate test cases for the manually developed tools, based on their specification

The soundness property for a code generator expresses that the execution of the model and of the generated code produces the same result. A key point is the definition of the «same» relationship between the input and output languages. The study that led to the technological choices and preliminary experiments on the scheduling of blocks in the data-flow models is described in more detail in [16].

5. Scope of the toolset

The Gene-Auto toolset covers a small part of the software development process. The toolset is intended to replace the programmer in the coding phase of a subset of control applications. However, usage of such tool influences the organisation of the whole lifecycle. As the analysis and design models must be eventually fed to an automatic tool for code generation, the models must be composed according to the "expectations" of this tool. To have the full effect of using an automatic tool, the testing processes should be adjusted to leave out the testing of those features that are already guaranteed to be correct by the code generator. The figures below illustrate the scope of the Gene-Auto toolset in the IEC 61508-3 V-model.

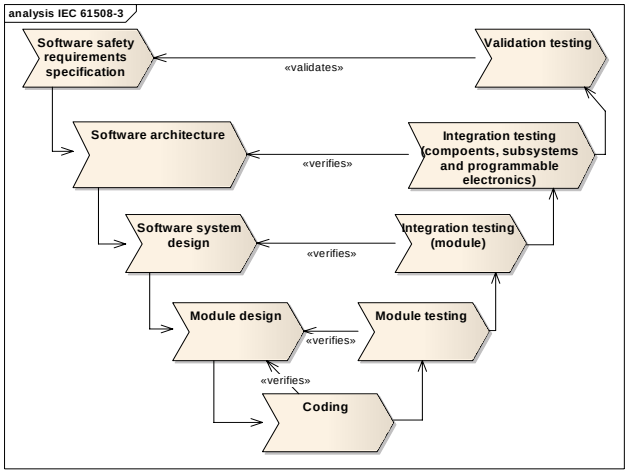


Figure 5: Simplified version of the IEC 61508-3 V-model

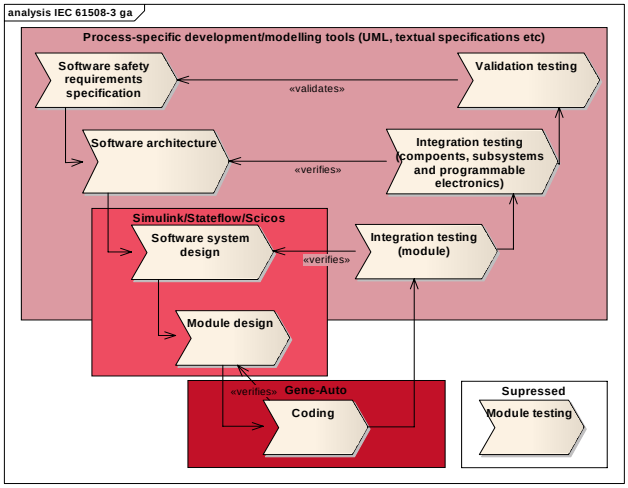


Figure 6: IEC 61508-3 V-model with suggested tools from the Gene-Auto toolset

Figure 5 presents the V model as suggested by the IEC 61508-3 standard. Figure 6 has the same model with annotations of Gene-Auto suggestions on tool usage in each phase. Gene-Auto does not have any effect on the first phases of the software development. The requirements specification and software architecture are composed using the procedures selected by the given organisation (a candidate for those phases of development from the open-source domain is the TOPCASED toolset). In system design the general architecture description remains with the tool that was originally used to gather the information. The behaviour of modules that will be later coded using the Gene-Auto ACG will be modelled using Simulink/Stateflow (or alternatively Scilab-Scicos) toolset. When entering the coding phase, the Simulink/Stateflow/Scicos models are converted to executable code using the Gene-Auto toolset. The responsibility of a programmer is to write the integration framework (communication middle-ware, hardware/ platform/ application-specific libraries etc) as required by the system architecture. The next phase – module testing – is suppressed for the auto-coded software in an ideal case. The functionality of the modules is verified on the model and the code generator guarantees a correct transformation from the model to software code. In a real process, it is still quite likely that some of module testing is necessary for aspects that can not be verified (or are too costly to verify) adequately on the model. For instance, numeric precision and implementation algorithms may be different in simulation and runtime libraries. In such cases a special unit test must be composed. The rest of the integration and testing process follows the rules set by the given organisation and is not directly influenced by the usage of the Gene-Auto toolset.

Another example of the development process illustrates the splitdown of work between the systems engineer and the software engineer. We take the suggested tool flow from the SCADE literature [17] as a basis and compare that to the proposed Gene-Auto process.

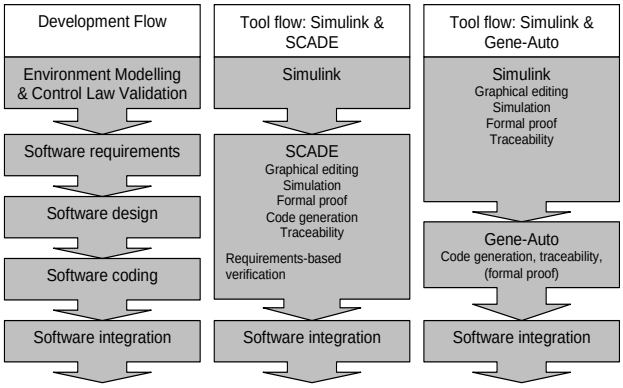


Figure 7: Comparison of tool flows suggested by SCADE and Gene-Auto

Figure 7 above illustrates the tool usage in software development process and also compares it to the SCADE as closest to the approach that Gene-Auto supports. The original picture is borrowed from [17] and modified to show Gene-Auto in the same context as SCADE. The first column in the figure describes the classical phases of software development process for developing control software. Traditionally the Environment modelling & control law validation is done by systems engineers and most likely in continuous time domain. The rest of the development activities are the responsibility of a software team. All models used to describe the behaviour of software must be discrete time models.

One of the most popular tools used by the system engineers is Simulink. The tool allows both discrete and continuous time models to be created, simulated and, using the design verifier (a model-checker), also formally verified. After the control law has been designed in continuous time using Simulink, it has to be converted to discrete time, refined to meet the requirements imposed by the software architecture and the implementation platform and, finally, transformed into executable program code. The approach suggested by SCADE suggests converting the model into a different formalism that is semantically a step closer to the final software implementation – a Lustre-based model in the SCADE tool. The SCADE tool can be then used to do further software design activities, formally verify the model with a model-checker and generate software code.

Whilst in general such explicit multi-step approach is reasonable for developing systems, the experience of Gene-Auto partners shows that there exists a wide range of applications where the intermediate

software model gives little or no value. In many cases the system model describes already adequately the algorithm to implement as well as the structure of the designed software. Additions during the software design step are minor and can be easily done on the original model created by systems engineer (provided that the software part of model is discrete-time model). Simulink supports validation of the model through simulation. A recently added proof-checker also allows formal verification of the model. The approach used in the Gene-Auto toolset proposes continuous refining of the model created by the systems engineer, until it is suitable for direct transformation into software code. This way a common modelling and simulation platform can be maintained throughout the design and (model) verification phases.

While providing a "one-step" approach from (annotated) systems model to software code for the engineer, internally the Gene-Auto toolset still uses intermediate models/languages for several reasons. First and the main reason is safety. It is widely known that since Simulink is a general-purpose simulation framework it allows composing models that are not suitable for direct implementations in safety critical environments (the models can be context sensitive, subsystems too closely coupled through global variables, usage of Stateflow allows to create infinite cycles etc.) While converting the Simulink model into an input model, the Gene-Auto toolset detects and reports back to the user the constructs that are not considered safe. Processing stops and can be continued after fixing the model. This is similar to the process in SCADE with one exception – the intermediate model is completely hidden from the user and will never be modified manually. All necessary modifications are made on the original Simulink model to assure that the original assumptions made by the systems engineer still hold and we do not alter the solution by adjusting the intermediate model. The second reason for an intermediate model is the wish to support input models from several model editors. During the first step the model in a source language is translated to one common language and all the other transformation steps are not dependent any more on the source tool that produced the model. Currently Simulink, Stateflow and Scicos models are supported.

It would be unfair to finish this chapter before commenting the relations between the Real-Time Workshop (RTW) [18] code generator that belongs to the Matlab product family and Gene-Auto. A reader who is familiar with RTW probably noticed already, that the approach suggested by Gene-Auto and that of RTW are very similar. This is no surprise, as process-wise the goal is the same – both tools are set to transform Simulink/Stateflow models into executable program code without producing any explicit intermediate models. The differences are in the level of control, openness and adaptability. Open source code, gives more possibilities for controlling the behaviour of the code generator than code templates provided by RTW, and contribute to vendor independent maintenance in case the support periods are long or the tool usage is too different from the mainstream. Gene-Auto can be adapted to support input models from different tools (it already supports Scicos). And finally, the open nature of the Gene-Auto toolset simplifies its qualification as a code generator for safety critical processes.

6. Test cases

Validation that the Gene-Auto tools meet the requirements set by the industrial partners in the project is taking place on various concrete test cases from following application areas:

- Aircraft flight control
- Automatic navigation
- Navigation display
- High reliability data transfer systems
- Servo control
- Car engine knock control strategy

The total number of validation test cases is 9 with an average of 1 500 blocks per model. The total LOC (Lines Of Code) of the manually written or automatically generated reference code is approximately 70 000.

7. Current status

The Gene-Auto project started in mid 2006. Its current progress can be summarised by the following:

- A Java-based Simulink code generator exists, supporting all major modelling constructs of the discrete part of Simulink and a selected set of standard and custom Simulink blocks.
- A Haskell-based Stateflow code generator exists that supports the full power of Stateflow models, including parallel automata, event broadcast schemes, graphical functions and truth tables.
- A Scicos Gene-Auto "palette" has been developed. It is available as a plug-in for Scilab.
- The integrated toolset has entered the first phase of industrial evaluation.
- A research team is working on the formal verification of the transformation process. For a part of the transformation there exists a prototype that has been developed with the Coq

proof system [19] and is based on the formal specification of the transformation.

8. Conclusions

We have described a work in progress on a code generator that is intended for real-time embedded systems in the safety critical domain. An assumption made by the project consortium is, that there exists a large enough subset of control applications, where the models created by a systems engineer can be transformed to executable code without composing explicitly intermediate software-specific models or additional specifications. The validity of this assumption is confirmed by the experience of the end users involved in the project who are using either commercial or in-house code generators for similar applications for a long time and with great success.

Commercial tools with similar functionality exist on the market already (SCADE from Esterel technologies, RTW from Mathworks, TargetLink from dSpace to name the biggest). All of them, except SCADE, lack one important quality that does not allow to get the full benefit from their usage in the development process of safety critical systems. A code generator employing the model-driven engineering approach can give savings in the development time from two interrelated phases. First, the time for coding is dramatically cut. Secondly, there are savings that come from testing – assuming that the solution was already verified on the model, some tests on the generated code can be suppressed. This is acceptable only if the code generator is qualified for that application domain and for the given process. The qualification effort is usually too high to be undertaken by a vendor of a general-purpose code generator. The closed nature of a commercial tool and lack of information about its development process does not allow the end-user to do the qualification either. The Gene-Auto tool development process supports the qualification of the toolset according to the highest industry standards (in particular the DO-178 avionic standard) and the consortium aims to prepare all necessary material for the qualification to take place in the user context.

From the viewpoint of qualification SCADE is an exception in the domain of code generation and modelling product. The disadvantage of SCADE for the chosen group of applications is that it binds the user to specific formalism. Powerful converters are provided to import models form tools such as Simulink, however before generating code the user is still required to work with the SCADE model. As pointed out earlier, our assumption was that for a certain group of applications the intermediate model is not necessary and restricts the engineer too much.

As a summary the distinguishing characteristics of Gene-Auto toolset are:

- Open source approach to facilitate long term support, adaptability and tool qualification
- Support for multiple input formalisms and fully automatic transformation from input model to executable code
- Explicit support for qualification through open architecture and suitable development process
- Formally verified components
- Optimised output code
- Support for optional verification, transformation and optimisation steps performed by external plug-in components

Early tests show, that the toolset is capable of generating code from industrial-grade models. Comparison of the generated code and code written manually by experienced programmers is currently ongoing.

9. Related work

Gene-Auto is related to several projects in the French “Aerospace Valley” cluster around the development of a toolset for safety critical real-time embedded systems:

- TOPCASED [13], funded by the French ministry of industry (FCE call) and the Midi Pyrénées regional authorities aims at building an open source CASE tool based on Model Driven Engineering for the left side of the V cycle. TOPCASED will rely on Gene-Auto for generating code for functional and automaton models. TOPCASED may provide tools around Gene-Auto modelling language.
- OpenEmbedd [20], funded by the French ANR (RNTL call), aims at integrating various existing projects around Model Driven Engineering for the design or real time embedded systems. OpenEmbedd offers graphical model editors for several languages, including UML2, SysML, AADL, SDL, SME, etc. It is based on the Eclipse platform and TOPCASED toolset. It also offers model transformation tools such as ATL and Kermeta, and model checking tools.
- SPaCIFY, funded by the French ANR (RNTL call) aims at defining the next generation of modelling language with a synchronous semantics and the associated CASE tools for on board space applications. SPaCIFY will rely on Gene-Auto for generating code for functional and automaton models. SPaCIFY will provide tools around Gene-Auto modelling language for model editing and checking.

- ES_PASS, an ITEA2 project around the use of static analysis for the V & V of source C code and target binary code will provide requirements for the generated code for improving the quality of the analysis by providing model level properties that are usually removed during code generation.
- SPICES and TWINS that are both ITEA projects around the architectural design and global co-design cooperate with Gene-Auto in order to have a coordinated development approach.

In the future, Gene-Auto could be integrated with the TOPCASED platform as a qualified back-end for generating source code from Scicos, some UML2 and SysML diagrams, SAM, SDL, etc.

Gene-Auto is related also to other projects and tools, which involve the use of synchronous languages for modelling safety critical embedded real-time systems, such as the SCADE Suite [5] (based on the Lustre [6] and SyncCharts [21] languages) from Esterel Technologies and PolyChrony SME [22] (based on the Signal [7] language) and to initiatives for defining a safe subset of Simulink/Stateflow such as [12]. The main difference is in the approach, we start from the whole Simulink/Stateflow and try to minimise the restrictions required to make it safe whereas these works started from existing safe languages and extended it towards something similar to Simulink/Stateflow (see, for example [23]). The Gene-Auto approach is therefore more user friendly, whereas the other ones are more minimal and clean on the semantic side.

10. Acknowledgements

Gene-Auto is an ITEA (Information Technology for European Advancement) project (ITEA 05018). The Gene-Auto Consortium involves following industrial partners: Airbus France, Barco, EADS-Astrium, Siemens VDO, Thales Alenia Space, software developers: IB Krates, CRIL Technology and academic institutions: ENSEEIHT, INRIA and Tallinn University of Technology. The authors want to thank the project members for their contribution.

11. References

[1] Schmidt, D. C.: "*Model-Driven Engineering*", IEEE Computer February 2006

[2] The MathWorks: "*Simulink® - Simulation and Model-Based Design*" http://www.mathworks.com/products/simulink/

[3] The MathWorks: "*Stateflow® - Design and simulate state machines and control logic*" http://www.mathworks.com/products/stateflow/

[4] INRIA: "*Scicos: Scilab's block diagram modeler/simulator*" http://www.scicos.org

[5] Berry, G.: "Esterel v7: From verified formal specification to efficient industrial designs" – Proceedings of the FASE'05 conference, 2005

[6] P. Caspi, D. Pilaud, N. Halbwachs, and J. Place "*Lustre: a Declarative Language for Programming Synchronous Systems*", Proc. ACM Symp. on Princ. of Prog. Langs. (Munich, West Germany), 1987

[7] Le Guernic, P., Benveniste, A., Bournai, P., Gautier, T.: "*SIGNAL - A Data Flow-Oriented Language for Signal Processing.*" – IEEE Transactions On Acoustics, Speech, And Signal Processing, 1986, ASSP-34 (2), 362-374

[8] Pouzet, M.: "*Lucid-Synchrone*" http://www.lri.fr/~pouzet/lucid-synchrone/

[9] Esterel Technologies: "*Esterel Studio*" http://www.esterel-eda.com/products/

[10] Esterel Technologies: "*SCADE Suite*" http://www.esterel-technologies.com/products/scade-suite/

[11] TNI Software: "*RT Builder Product Overview*" http://www.tni-software.com/fr/produits/rtbuilder/index.php

[12] Scaife, N. Sofronis, C., Caspi, P., Tripakis, S., Maraninchi, F: "*Defining and Translating a "Safe" Subset of Simulink/Stateflow into Lustre*" – Proceedings of the 4th ACM international conference on Embedded software. 2004, 259–268.

[13] "*TOPCASED Toolkit In OPen source for Critical Applications & SystEms Development*" http://www.topcased.org

[14] Caspi, P., Pouzet, M.: "*Synchronous Kahn networks*" In ICFP '96: Proceedings of the first ACM SIGPLAN international conference on Functional programming, pages 226–238. ACM Press, 1996.

[15] Harel, D: "*Statecharts: A visual formalism for complex systems*" – Science of Computer Programming, 1987, 8 (3), 231-274

[16] Izerrouken, N. Thirioux, X. Pantel, M. Strecker, M: "*Certifying an Automated Code Generator using formal tools: Preliminary experiments in the Gene-Auto project*" – Proceedings of the ERTS'08 conference (Toulouse, France), January 2008

[17] Esterel Technologies: "*Simulink Users Connect with Esterel's SCADE Suite for Safe Embedded Software*" http://www.esterel-technologies.com/files/Simulink2SCADE.zip

[18] The MathWorks: "*Real-Time Workshop®*" http://www.mathworks.com/products/rtw/

[19] "*The Coq proof assistant*" http://coq.inria.fr/

[20] OpenEmbeDD: *"Model Driven Engineering open-source platform for Real-Time & Embedded systems*" http://openembedd.org/home_html

[21] André, C: “*SyncCharts: a Visual Representation of Reactive Behaviors*”, I3S Research Report # 96.56, Sophia Antipolis (F), April 1996.

[22] Brunette, C., Talpin, J.-P., Besnard, L., Gauthier, T: "*Modeling multi-clocked data-flow programs using the Generic Modeling Environment*". Synchronous Languages, Applications, and Programming (SLAP'06). Elsevier, March 2006.

[23] Colaço, J.-L., Hamon, H., Pouzet, M.: “*Mixing Signals and Modes in Synchronous Data-flow Systems*” ACM International Conference on Embedded Software (EMSOFT'06), Seoul, South Korea, October 2006.

8. Glossary

MDE: model-driven engineering

ACG: automatic code generator