Towards Reliable Code Generation with an Open Tool: Evolutions of the Gene-Auto toolset

A. Toom[1,2], N. Izerrouken[2, 3], T. Naks[4,5], M. Pantel[2], O. Ssi Yan Kai[3]

1: Institute of Cybernetics at Tallinn University of Technology, Akadeemia tee 21, EE-12618 Tallinn, Estonia
2: IRIT-ENSEEIHT, Université de Toulouse, 2, rue Charles Camichel, 31071 Toulouse Cedex, France
3: Continental Automotive France SAS, 1, avenue Paul Ourliac, 31036 Toulouse Cedex, France
4: IB Krates OÜ, Mäealuse 4, 12618 Tallinn, Estonia
5: Department of Computer Control of Tallinn University of Technology, Ehitajate tee 5, EE-19086 Tallinn, Estonia

Abstract: This paper presents the current status of the Gene-Auto[1] automatic code generator, an open source tool for safety critical embedded systems and thus to be qualified according to the DO178/ED-12 avionic software certification standard. Gene-Auto transforms Simulink, Stateflow and Scicos models to MISRA C and Ada SPARK code. The paper focuses on the second version of Gene-Auto and the changes since the first version presented at ERTS'08 [1], [2]. We will also summarise the development process, where a classical approach has been mixed with formal specification, development and verification of some of the toolset components using proof-assistants. This development process has led to preliminary positive feedback from the avionic certification authorities. The toolset has also been evaluated in a number of industrial test cases from the avionic, automotive and aerospace domains, proving that it is a mature prototype, which can be considered for industrial projects in near future. We present recent additions to the toolset, like generating Ada SPARK source code, adding an EMF based interface, improvements in the block sequencer algorithm developed with a proof-assistant and some details on the qualification aspect of the toolset. We also mention some industrial feedback on Gene-Auto.

Keywords: Automatic Code Generation (ACG), Certified systems, DO-178, ED-12, Formal verification, Proof assistant, Coq, Open-source, Simulink/Stateflow, Scicos, Ada SPARK, MDE, EMF.

1. Introduction

Domain specific languages and visual modelling formalisms have largely replaced natural language specifications and chalk drawings in the systems engineering domain. Software tools are used to create, analyse, verify and transform formal and semi-formal specifications until they can be executed on high-performance specific hardware. Open formats and open tools are expected to bring long-term stability and flexibility to the maze of languages and software needed to manage the transformation of a high level functional specification to optimised machine code. The Gene-Auto project was set up to answer the needs of ever-increasing complexity, code size, safety and long-term maintainability requirements and develop an open source code generator from the widely used Simulink/Stateflow[2] modelling languages and their open source counterpart Scicos[3] to a general purpose programming language like C, as an open tool with open architecture and intermediate languages, but also fully adaptable and qualifiable according to the needs of the safety critical transportation domains (regulated by e.g. DO178/ED-12 in the avionics).

Previously [1] we have presented the Gene-Auto architecture, its input formalisms, intermediate languages and role in the overall embedded systems software development process. In [2] we gave an overview of the early experiments with formal methods on parts of Gene-Auto together with a comparison of the suitability of different formal methods for certifying an ACG (Automatic Code Generator). In Chapter 2 of this paper we give a short overview of the Gene-Auto toolset. Chapter 3 explains the scope of Gene-Auto in the embedded systems' design process. Chapter 4 describes an EMF based interface added recently to Gene-Auto to facilitate interoperation with other tools. In Chapter 5 we describe the addition of the Ada SPARK language output to the Gene-Auto toolset. Chapter 6 gives the main improvements in the formal methods based development of a part of the Gene-Auto toolset. In Chapter 7 we discuss some aspects related to the qualification of such a tool. In Chapter 8 we mention some user feedback and Chapters 9 and 10 outline some future and related works respectively.

2. The Gene-Auto toolset

In this chapter we give a brief overview of the Gene-

This work was partly funded by national authorities through the ITEA project Gene-Auto and the ITEA2 project OPEES

1 www.geneauto.org

2 Part of the Matlab toolset from The MathWorks Inc. www.mathworks.com

3 Part of the SciLab toolset developed at INRIA and managed by the SciLab Consortium. www.scilab.org

Auto toolset to facilitate the understanding of the topics addressed in the subsequent chapters. The Gene-Auto toolset takes as input models in various modelling formalisms: Simulink block diagram, Scicos block diagram, Stateflow chart, Stateflow graphical function and Stateflow truthtables. The first two formalisms are somewhat similar to the classical dataflow models that contain nodes doing computations and data flowing between the nodes. However, in the case of Simulink the formalism contains also a special "function-call" triggering mechanism that adds imperative scheduling to the data-flow model and thus creates a unique complex formalism allowing to express system engineering aspects that are sometimes hard to express in pure dataflow formalisms. A Stateflow chart is similar to the classical StateCharts formalism, but again, it adds several unique constructs that make it on one hand a very powerful language, on the other hand also more complex and potentially more error-prone. Graphical functions and truthtables are imperative sublanguages of the Stateflow language that are internally implemented as imperative flowcharts.

The whole transformation chain from the input models to imperative target code is split into several elementary tools performing one or several related model transformations to isolate functionally independent parts of the toolset from each-other and facilitate the writing of detailed specifications for each model transformation, as well as testing and validating them. This approach gives both, the flexibility to add/replace functional modules and a fine grain view over the toolset to write detailed module level requirements and perform module level validation needed for qualifying the tool. There are two intermediate languages in Gene-Auto. First, all input models are converted to the GASystemModel language, which is semantically close to the input formalisms, but is independent from the concrete representation of the input models. This model is step-wise refined and finally converted to the GACodeModel language. A model in the GACodeModel language is further refined and then converted to specific target code (C or Ada, currently).

In total there are 12 elementary tools implementing 9 major transformation steps from the source model to target code. The sequence of transformations begins with an initial importing step depending on the input language syntax and ends with a printing step specific to the particular target language. For the first step there exists a separate importer tool for each supported input language (currently, Simulink, Stateflow and Scicos). Likewise, each target language has its own printer component taking care of adapting the generated code model to the specifics of this language and finally printing out the target code.

Gene-Auto supports a subset of the discrete part of Simulink/Stateflow and Scicos. Its standard library supports currently around 40 Simulink native blocks, 20 custom blocks and 40 Scicos blocks. Both single and multirate models are supported, as well as explicitly triggered dataflow models ("function-call" triggered models in Simulink). All main Stateflow constructs are supported, except for the local event broadcast. The semantics of the local event broadcast that has a run-to-completion semantics in Stateflow and UML is undesirable in critical embedded systems. Instead, Gene-Auto supports a more synchronous style of modelling of statecharts. External broadcasts made to the outside of the statechart have an important role in the scheduling of the combined Simulink/Stateflow models and are supported.

3. Scope of Gene-Auto in the system development process

Figure 1 presents a typical V-cycle of system engineering used in the process of developing most of the complex systems with high requirements. Including safety critical and high integrity systems. The left branch captures system definition in a sequence of successive refinement steps. For each project definition step there is a corresponding verification or validation step on the right side. Also, each subsequent refinement step verifies some aspects of the previous one. Design of complex systems involving software usually begins with one or many stages defining first the user requirements for the end-system in different level of detail, then the software specification and implementation and finally ends with acceptance testing of the entire system performed by the end-user or client.

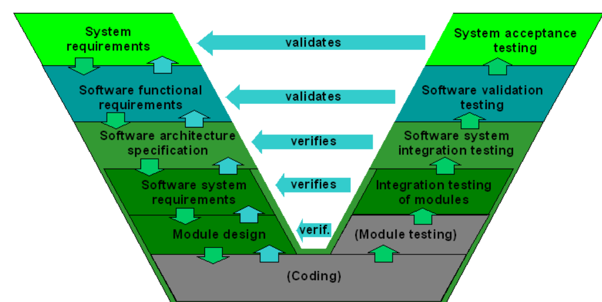


Figure 1 The common V-cycle of the system design process.

The main intended scope of Gene-Auto is in the lower part of the V capturing the design loop that involves the software system requirements specification, module design, implementation and the integration testing of modules. Defining the target system as a model containing all necessary

implementation details gives a possibility to replace the manual coding step with code generation. The code generation is performed by Gene-Auto. Provided that the code generator tool is reliable enough for the particular purpose (e.g. in the avionic industry it meets the DO-178/ED12 requirements for a development tool of the relevant criticality level) the module testing can be omitted. This is based on the assumption that such modules have been already verified in the previous design step either by simulation, testing or formal verification. If the source model contains several logical or physical modules and captures their interaction, then the functional testing of these modules' integration can also be reduced or even eliminated. It must be noted, however, that while the DO-178B(C)/ED12-B(C) qualification has been a crucial initial goal for the Gene-Auto toolset and its development follows the guidelines of these standards, it is not qualified yet and a lot of the qualification activity is still to be done.

Secondly, Gene-Auto was conceived as an open tool, whose platform can be used for various design and verification/validation steps in the MDE process. The code generation architecture of Gene-Auto consists in a sequence of model transformations in itself. The individual model transformations can be freely combined, interleaved with other tools etc. The models can be exported to external tools and reimported at any chosen stage. Thus, it is possible to complement the code generation steps with e.g. custom optimisation, automatic or semi-automatic verification etc. Alternatively, it is possible to inspect and/or modify the intermediate models with external model editors, simulate, verify and refine them and finally use Gene-Auto as a code generation backend. Chapter 4 describes one way to implements such tool interaction using the EMF (Eclipse Modeling Framework) and gives some examples.

The value and scope of automatic code generation can be further increased, if the functional application model is complemented with the non-functional target platform constraints. One approach combining domain specific architectural and functional modelling is being developed within the SPaCIFY project[4]. Chapter 4 describes briefly the linkage between Gene-Auto and SPaCIFY models.

Models like those supported by Gene-Auto occur also at higher levels of the V-cycle than the systems' software development. For instance, statemachines and flowcharts can be used to specify high-level functional requirements of the systems' behaviour in a more formal way. The final application needn't be directly derived from these models. Instead, they can be later used as a reference for validating the implemented software or complete system. A language specifically designed for this kind of behavioural modelling and analysis for critical embedded systems is TOPCASED-SAM (Structured Analysis Model)[5]. Gene-Auto support of the TOPCASED-SAM language is going to be implemented using the approach described in Chapter 4.

Finally, Gene-Auto can be used as an alternative code generator, an oracle, to test the output of other code generators or manual coding of higher level requirements expressed as models. Such an approach has been followed in the EDONA project[6]. Within this project the Agatha tool has been used to analyse the C code generated by Gene-Auto and generate test cases based on it. The EDONA platform has been described in more detail in [3] and [4]. Besides this pure integration of tools, cooperation with EDONA has motivated also addition of new features to Gene-Auto, like support of parameterised subsystems and likewise also influenced the set of modelling guidelines used within the EDONA projects (e.g. requiring more precise type specifications).

4. EMF-based metamodel and interfacing with external tools

There exist many powerful frameworks dedicated to metamodelling, domain specific language (DSL) design, transformation specification, validation, etc. Several of them are directly inspired from the Object Management Group's (OMG) Meta-Object Facility (MOF) standard, for example, the Eclipse Modeling Framework (EMF)[7] and Kermeta[8]. The EMF provides facilities for modelling and code generation for building tools and other applications based on a structured data model. It is used to implement several important components of the open source Eclipse platform itself as well as commercial applications. The Eclipse platform provides also powerful components for graphical modelling, data handling, general tool design, etc. Hence, also several projects related to model based design for embedded applications, like TOPCASED, SPaCIFY and PolyChrony SME have been implemented on the Eclipse platform.

However, powerful generic frameworks meant for a large user-base like EMF provide a challenge for developing qualified applications with very high qualification requirements on the software. Typically, such frameworks are quite complex and introduce to the application code dependencies on parts of the framework code. To satisfy the DO-178B/ED12B

[4] http://spacify.gforge.enseeiht.fr/index.php

[5] http://www.topcased.org
[6] http://www.edona.fr
[7] http://www.eclipse.org/modeling/emf
[8] http://www.kermeta.org

requirements, such code has to meet the same strict verification and quality requirements, as the main application code. Isolating and verifying separately the needed functionality of the framework code from the unneeded one can be a considerable task. It can be less costly to implement the specific required functionality from scratch or with more lightweight methods and to qualify only that. The latter approach has been taken for developing the core of Gene-Auto. On the other hand, to be able to simply interface and interact with the many EMF based modelling, simulation, verification and other applications from relevant domains an alternative approach has been designed and is described below.

The main part of the Gene-Auto toolset is implemented in Java. At its core are the GASystemModel and GACodeModel language implementations. The structure of these languages is modelled in an UML CASE tool (Enterprise Architect). From these UML models Java classes are generated. All the functionality that is required to manipulate models conforming to these languages is implemented manually. A separate ModelFactory component is implemented to serialise the models from memory to a simple XML based file format and vice versa. The core toolset uses mostly only basic Java, a rather limited subset of an XML library (Xerces) and a parser generator library (AntLR).

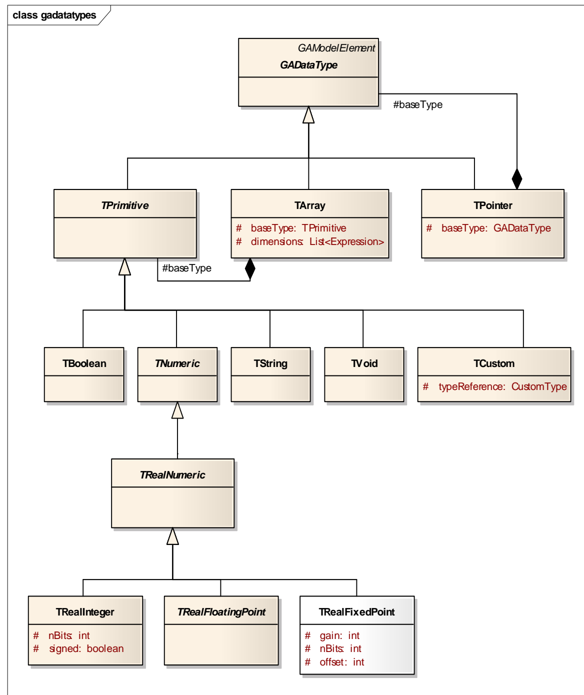
Parallelly to the Gene-Auto main branch there is an Ecore (the EMF metamodel of metamodels) version of the GASystemModel and GACodeModel language metamodels generated from the UML models. This Ecore metamodel is central to any other activity involving the EMF. Among other things it can be used to generate a Java implementation of the metamodels/languages. This implementation is intrinsically relying on and capable of taking advantage of the EMF's built-in functionalities for model manipulation, serialisation, etc. It is also possible to generate a simple tree-view editor of the language from the metamodels with no extra work. A screenshot of the generated tree-view editor is displayed on Figure 3.

Thus there exists a special purpose compact language implementation that is used in the main branch of the toolset and a separate structurally equivalent EMF based language implementation. Conversion between these two language implementations is done via specialised Model-Factory components. These components convert the model instances from one format to another. In practice, these components are bundled into two elementary tools: `gaxml2ecore` and `ecore2gaxml` converting XML files from the Gene-Auto native format to the EMF based format and vice versa. Combining these tools and external EMF based applications with the main Gene-Auto model transformation and code generation steps it is possible to derive many different toolchains with quite different functionality and/or implementation. Such toolchains can be used in the many applications that have weaker requirements than the DO-178/ED-12 qualified development tools. In case the external tools only perform complementary verification tasks and do not change the output code, they could also be qualified as verification tools in the DO-178/ED-12 sense, for which the qualification requirements are significantly less strict.

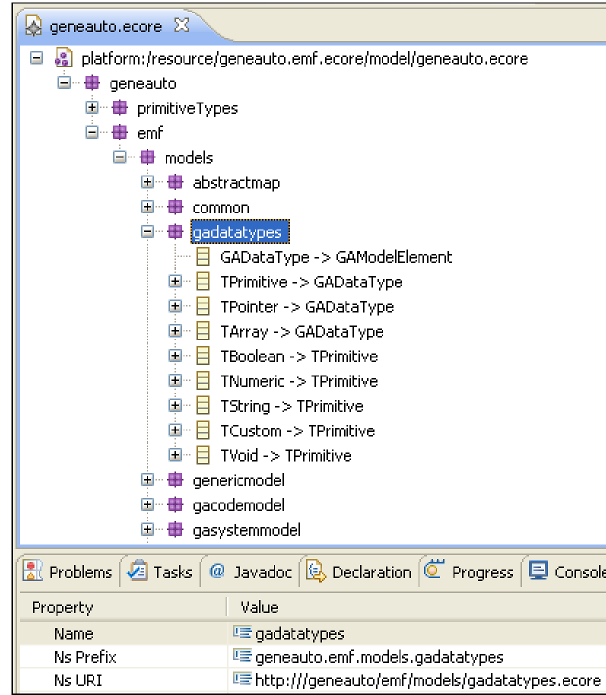
The EMF based interface of Gene-Auto is already being used within other projects. Two of them are SPaCIFY and Polychrony/SME. The SPaCIFY project is combining the architectural and functional aspects of embedded systems modelling. At its heart is the Synoptic language [5]. Linking Gene-Auto and SPaCIFY amounts to defining a mapping between the functional part of the Synoptic language and the GASystemModel languages. Technically, the model transformation is implemented using the ATL language[9]. Within the Espresso team a translation is developed between the Signal synchronous language based modelling environment Polychrony/SME[10] and Gene-Auto. Besides the model editing and simulation capabilities the Polychrony/SME suite also offers powerful means for model analysis and verification. Such analysis could, in principle, be used as an intermediate step in Gene-Auto to perform stronger dataflow optimisations.

Last, but not least, the EMF based language metamodel can be directly used for specifying the functional requirements of the Gene-Auto code generator itself. First, the EMF contains a validation framework that can be used to check conformance of model instances vs. the metamodel and additional formal constraints expressed e.g. in the standard Object Constraint Language (OCL). Secondly, model transformation steps can specified in a model transformation language like ATL and the transformation instances can be automatically verified with Eclipse based tooling. Such approach has been already partly investigated and is going to be developed further.

[9] http://www.eclipse.org/m2m/atl/
[10] http://www.irisa.fr/espresso/Polychrony/

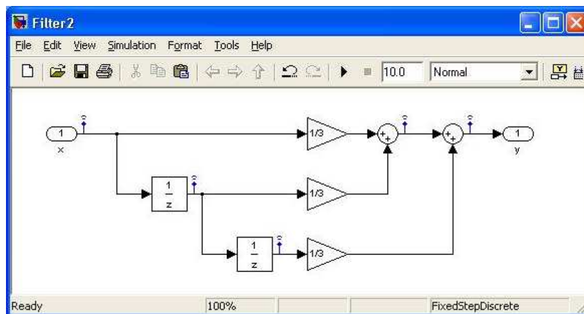


a

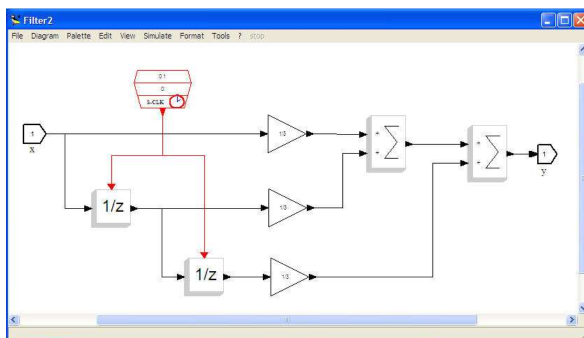


b

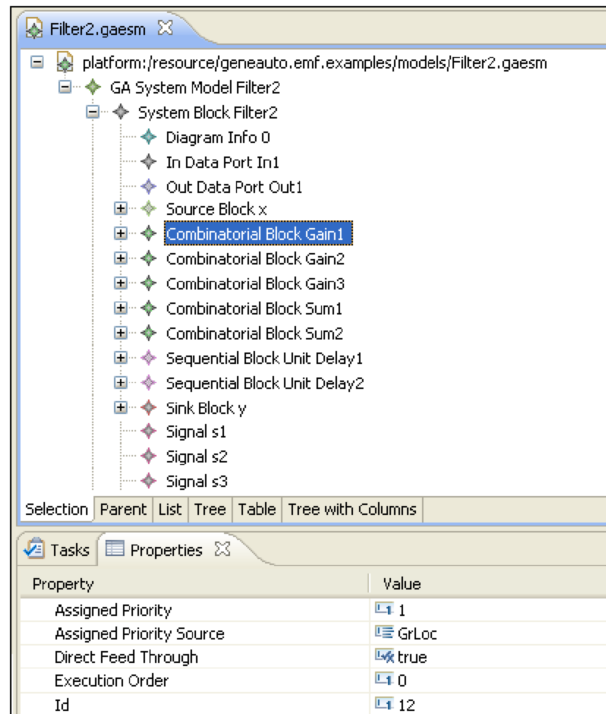
Figure 2 Fragment of the Gene-Auto UML metamodel (a) and the corresponding fragment from the Ecore metamodel (b)



a



b



c

Figure 3 A Simulink model (a), similar Scicos model (b) and a corresponding Gene-Auto model in the EMF based GASystemModel tree-view editor (b)

5. Adding Ada SPARK output to Gene-Auto

One of the aims of defining a modular open architecture was to facilitate adding additional input and output languages to the toolset. Ada is a programming language that is specifically designed to suit the needs of critical embedded and real-time systems. It features strong static typing, static and run-time checking of many kinds of errors, a package system, object oriented programming, exception handling, parallel tasks, etc. SPARK [6] is a language based on a restricted subset of the Ada language coupled with an annotation language allowing the programmer to specify formal requirements about the behaviour of the program. There exist also tools for static verification around SPARK[11] that allow to check the absence of general run-time errors like numerical overflow or division by zero and that the user specified properties hold. The proofs will either be generated automatically or developed with the programmer's assistance for the more complex cases. Thus the Ada SPARK language is often used in the development of very high criticality systems.

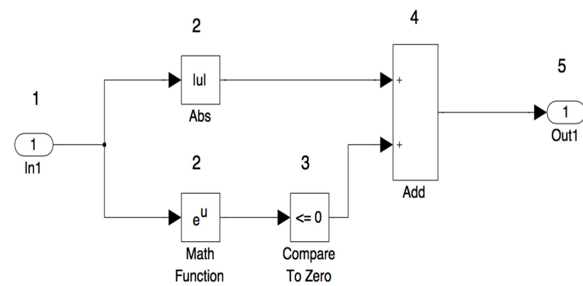
In the first phase it was decided to add SPARK compatible Ada language output to the Gene-Auto code generator. Generating SPARK formal annotations about the properties of the model/code was postponed to a future extension. As the GACodeModel language is an abstract imperative language with common constructs found in most programming languages it was not a very big conceptual step to add the SPARK language output to the existing C language output. However, as Ada, and especially SPARK, are more precise and restrictive than Simulink and C, additional work had to be done to make the Gene-Auto type system and the generated code compatible with SPARK. For instance, in Simulink and C it is legal to assign integers to floating point data and even floating point data to integers, while in Ada and SPARK it is not. Moreover, an integer value from a larger range type can be assigned to a variable with a smaller range type in both Simulink and C. Again, in Ada and SPARK it is not possible. In order not to enforce the user to make all the type conversions explicit in the input model, such automatic type conversions were added to the Gene-Auto Ada backend. This is, of course, a somewhat debatable point: there are those, who want the code generator to support as much as possible from the features of the input language and those, who say that the input model should be a precise software model with explicit type information etc. Here, a pragmatic decision was taken to support more input models and add the casts required for Ada SPARK automatically. Note, however, that the aspect of arithmetic overflow is ignored, justified by the fact that the absence of overflow can be checked either with Simulink verification tools on the input model or with other tools directly on the generated code (e.g. SPARK tools for Ada[12], Frama-C[13] for C) and the code generator tool should not be complicated with verification unrelated to code generation.

11 http://www.adacore.com/home/products/sparkpro/

6. Usage of formal methods

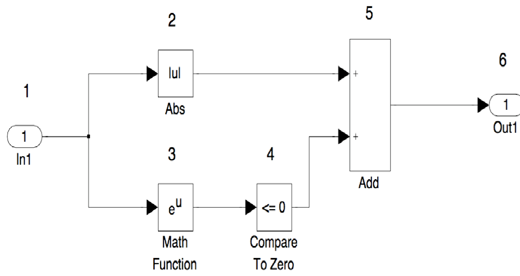
6.1 Specification and implementation of selected model transformations with Coq

We presented in [2] a first block sequencer for Gene-Auto developed using the Coq proof assistant. The first developed solution produced partial execution schedule for each block in the model based on the data-flow causality (see Figure 4). This partial ordering was refined using user provided block priorities and the block graphical position as done in Simulink. But, it reduced the potential concurrency expressed in the model. For instance, let us consider the blocks with identical execution order such as CompareToZero and Abs illustrated in the figures (a) and figure (b), the block Abs cannot be evaluated before the block CompareToZero, if the later has higher priority because its initial execution order is lower. Let us note that this algorithm also did not handle the "function-call" kind of control-flows. This initial work required a significant adaptation in order to allow all concurrent executions and to handle "function-calls", which are widely used in Simulink by industrial end-users to manage side-effects (environment input/output and memory management).



a. Partial preorder

12 As the expression on the right side of the assignment is being explicitly cast to a suitable type, then overflow isn't normally checked for the whole assignment, but it is still checked for the subexpressions of the right side.
13 http://frama-c.cea.fr/



b. Total preorder

Figure 4 Simulink data-flow model with a conflicting execution order.

A new version of the algorithm with execution dependency computation was proposed in [7] to take into account the mixing of data and control flows, by identifying all the blocks that need to be computed before a given block and the ones that must be executed before the outputs of the block can be used. This set allows to inherit the execution priorities along the data and control flows and thus to produce a correct total sequencing that fits best the end users' and semantic constraints. The new Block Sequencer tool containing this algorithm was integrated into the toolset and as a result the toolset was able to handle some real-life industrial test cases that previously was not possible.

There is ongoing work on extending the approach used in the Block Sequencer to be used in other model transformations in Gene-Auto. As an example, a similar model traversal can be used while inferring the types of untyped ports and signals in the model. The work of refining and formalising the Gene-Auto type system and typing algorithm is currently in progress.

6.2 Integration of formally developed elementary tools into the classical development process

In order to integrate the parts that were developed using the Coq proof assistant into the Gene-Auto toolset that is mainly developed in Java a specific process was designed. It was mainly constrained by the need to assess this process with respect to the certification authorities. The different elementary tools and the block library of Gene-Auto exchange data through XML files and a model reader and writer library that will be qualified using classical techniques. In order to avoid the development of a similar component with a proof assistant, we have chosen to develop Java front- and back-ends that translate the XML format to a very simple regular language for which a very simple converter was written and that is verified and will be qualified by independent proof-reading and unit testing. Then we developed a CaML wrapper that reads this input model as a text file and builds the data structures used by the CaML code generated by the Coq proof assistant. The CaML code is verified by independent proof-reading and unit testing with respect to the regular language and the Coq specification. This was considered a satisfying approach by the certification bodies. However, a more satisfying approach requires the use of a qualified formally verified Coq toolset that is currently partly being developed. See e.g. [8], [9] and [10] for some of the related works.

Similarly to the classical development process for the Java parts in Gene-Auto, the Coq specification is written with respect to the classical natural language user and tool specifications and verified by cross-reading. The implementation is specified as functions in Coq and proven formally correct with respect to the translated requirements and the functions are translated to CaML.

7. Qualification concerns

The final goal of the Gene-Auto development is to complete all the work required by the DO-178/ED-12 avionic software certification standards to qualify the toolset as a development tool. Qualification activities during the first phase of the project, that was completed in the end of 2008, concentrated on the planning activities (tool development, verification and (pre-)qualification plans), user requirements and formal specification and verification experiments. These elements were extensively discussed with the French CEAT certification authorities for early advices. Positive feedback was provided with respect to these documents acknowledging a sensible and adequate process and a good management of the difficulties. Full verification activities were then intentionally postponed as well as part of the requirements definition and detailed design documentation. Similar work is carried out in parallel in the ES_PASS ITEA2 project and will be continued in the OPEES ITEA2 project with thorough experiments around Gene-Auto and Frama-C.

The Gene-Auto project partners continue working on the qualification documentation and on solutions to simplify the integration of community developments and formal methods based work into a DO-178/ED-12 regulated development process. Under the umbrella of the Open-DO[14] framework some initial experiments on an open environment for requirement management and toolset testing have been taken. The aim is to make requirement management process lighter when compared to the current document-centric approach and allow seamless integration of the verification activities.

Qualification in the DO-178B context is known to be massive work and the whole process is sensitive to

14 http://www.open-do.org

changes in the software. A smallest change in the developed software or its requirements may trigger very large amount of verification activities to be repeated, unless one is able to prove the exact limits of the impact of the change. The way chosen in Gene-Auto is to divide the toolset into smaller isolated elementary tools. This way it is easy to demonstrate that a change in one transformation does not influence the others and the additional verification activities need to be done on the changed component only. While effective in terms of partitioning the development artefacts, this approach creates additional complexity with several levels of requirements. The *toolset operational requirements* describe the tool's end-to-end behaviour taking into account the relations between the input model and generated code. These high-level requirements are further split down into *elementary tool requirements* describing the behaviour of each transformation step. Based on the latter, the design of each elementary tool is derived and the software is developed. Traceability must be maintained throughout all the levels of specification and each level needs to be verified. This extra complexity is the price of the flexibility achieved by isolating the transformation steps.

From the process viewpoint two approaches are investigated in parallel to guarantee a sound transformation from model to code: (a) a formal-proof based approach described in Chapter 6 and (b) a classical development process coupled with test-based verification. The approach based on formal proofs has shown good results when developing one of the components. However, there are some concerns regarding the scalability of the approach (it takes considerable effort to define requirements with sufficient formal rigour). Also, the effort to go through the full approval by the qualification authorities is estimated to be high, given that such process has not been used for tool qualification before. Classical development process follows the known path in terms of qualification. Here the bottleneck is the amount of verification activities.

Regardless of the development process chosen for final qualification of the toolset a major challenge is managing consistency of requirements and guarantee adequate amount of verification. The path investigated currently is to integrate the requirement management and verification into one compact framework. Keeping the requirements in a system where all levels of specifications are interlinked allows determining impact of changes very quickly and necessary modifications can be done to all related artefacts at once. Dynamically interlinking the requirements with associated tests allows to determine the scope of required verification activities and execute large part of them automatically. Two specification levels (elementary tool requirements and design requirements) for the Ada printer component are defined in this fashion. The experiments were done with slightly modified instance of the FitNesse tool (www.fitnesse.org). The results show that using an environment maintaining hierarchical structure of requirements improves significantly consistency between the different levels. However, to scale up for full set of requirements, the tool support must be enhanced. Node interlinking mechanism of FitNesse is easy to use and intuitive, but more complex liking system with additional semantical information is required to handle large amount of requirements. The project team has also developed a testing framework allowing to automate the verification activities. This framework is currently used as part of the continuous integration environment and will be integrated with the requirement management in the future.

8. User-side experiments

During the first phase of Gene-Auto development (from 2006 to early 2009) the toolset was evaluated on 7 industrial case studies from the automotive and aerospace industries. Two academic case studies were performed by the Tallinn University of Technology demonstrating the usage of toolset in development of the navigation system of an autonomous vehicle. Additional studies have been performed by different parties after that. Results of two case studies performed by Astrium Satellites and Airbus in the context of the OBSYS project are presented in [15]. The overall feedback of these experiments is that Gene-Auto is a mature prototype that can be used also on real-sized industrial projects. The open platform of Gene-Auto has proven strong enough for rather complex applications and also flexible to allow user specific additions and optimisations. In some cases there was additional work required to make the existing models compatible with the functionality supported by Gene-Auto. E.g. Gene-Auto does currently not support the Embedded Matlab textual language (EML) that can be used to complement the graphical blocks of Simulink. In such cases the EML blocks had to be converted to black boxes containing C-code (called S-functions in Simulink) that were obtained by using The MathWorks' Real-Time Workshop (RTW) code generator. Adding support for the EML in the future was pointed out as a highly useful addition. In general, the case studies confirmed the functional equivalence of the code generated by Gene-Auto with respect to the reference code obtained by other established industrial tools.

9. Future work

As the user-side experiments show Gene-Auto can be used in applications of significant size and complexity. However, for a wider usage the supported functionality should be extended further, especially, the number of library blocks should be increased. Given, that the core technology of Gene-Auto is rather stable, adding more standard blocks to the library is a feasible task with little affect on other components. In principle, most of the commonly used blocks can be constructed by combining other smaller and simpler blocks. The size of the supported block library is a question of finding a good compromise between the development and qualification efforts and user-friendliness. A second functional addition that would be highly appreciated by many modellers is support for a textual action language like the Embedded Matlab Language that is available in Simulink. This addition, as well as extending the block library has been planned in the follow-up projects of Gene-Auto.

A separate path to extend the functionality will be to define an UML/SysML and MARTE profile for mapping the Gene-Auto semantics. This will allow to better integrate Gene-Auto with standard modelling tools, to edit and analyse the intermediate models and to import/export them from/to standard UML models. The work carried out within the SPaCIFY project will establish a link between the functional modelling that is the main focus of Gene-Auto and architectural constraints modelled with AADL.

There will be also ongoing work on the formal specification of the entire model transformation chain of Gene-Auto. We have generalised the work described in Chapter 6 and developed a generic framework to specify also other transformation steps like type inference/checking and code generation. Additionally, we plan to investigate a different more lightweight formal verification approach consisting of separately verifying the correctness of code generation with respect to its requirements and the semantic verification of the requirements.

An important work in parallel with the maturing and extending of the toolset is the refinement of the requirements of the core functionality, designing and performing tests and preparing the rest of the qualification data as described in Chapter 7.

10. Related work

Gene-Auto is related to many European and French projects around the development of tools for safety critical systems. Most importantly TOPCASED, SPaCIFY, ES_PASS and OpenEmbedd. Gene-Auto is also related to the synchronous language community of modelling safety critical embedded real-time systems such as the SCADE Suite[15] (based on the Lustre language) and PolyChrony/ SME (based on the Signal language). However, the difference from these is that Gene-Auto semantics is based on the Simulink/Stateflow semantics that cannot be fully mapped to the synchronous dataflow languages in a simple and traceable manner.

The MathWorks' RTW Embedded Coder[16] is a C code generator belonging to the Matlab/Simulink toolsuite that is quite powerful and widely used. However, while it allows some user-side customisation it is a closed proprietary tool and not qualified in the DO178/ED-12 sense. The open-source Scilab/Scicos toolset contains also a C code generator. Like RTW it does not generate Ada code and unlike the Gene-Auto toolset and RTW it generates a function call for each block. This doesn't allow carrying out optimisations during the code generation and the code requires potentially more stack space at runtime. The code generation functionality is also integrated with the rest of the toolset's functionality, making it thus hard to qualify or customise.

On the formal side there are important related works that rely on compiler verification and validation [12], [13], but the verification is focused on instances of compilation. A promising approach consists in the formal development of a correct-by-construction compiler, e.g. [14]. Our current work is based on this later approach. However, there is a significant difference with our proposal. In order to avoid departing from the usual industrial approach of qualification and ease its acceptance by the certification authorities, we do not work at the semantic level directly. Infact, we developed a two-step approach: in the first step, we translate the natural specification of requirements into a formal specification using the Coq proof assistant; then in the second step, these requirements are proved correct with respect to the semantics of the languages.

11. Conclusions

We have presented the development of an open toolset designed for critical embedded applications. The evolution of Gene-Auto is on one hand motivated by the benefits of having an open customisable architecture to allow first, the generic handling of similar modelling formalisms as well as target languages, but secondly also different kinds of tool interoperation on various system design and implementation levels. On the other hand, the evolution of Gene-Auto is constrained by having a small and robust core that is qualified primarily according to the DO-178/ED-12 standards, but also

[15] http://www.esterel-technologies.com/products/scade-suite/
[16] http://www.mathworks.com/products/rtwembedded/

other domain specific safety and quality standards used in e.g. space and automotive industries. The initial user-side experiments have shown the usability of Gene-Auto in an industrial environment. The growing number and complexity of requirements for embedded software, together with the multitude of different development and verification tools used in the design process only increase the need for open and reliable production tools.

12. Acknowledgements

Gene-Auto grew out of the ITEA (Information Technology for European Advancement) project ITEA 05018. The original Gene-Auto Consortium involved the industrial partners: Continental Automotive France, Airbus France, Barco, EADS-Astrium, Israel Aerospace Industries, Thales Alenia Space; SMEs: IB Krates, Alyotech, and academic institutions: INPT-IRIT/Université de Toulouse, INRIA and Tallinn University of Technology. After the project's end in December 2008 Gene-Auto has been developed further with the support of some of the former Consortium members and follow-up projects are being set up. The development of the Ada SPARK backend was funded by AdaCore and jointly performed by AdaCore and IB Krates. The work on EMF is funded through the ITEA2 OPEES project. The authors want to thank all the contributors of the Gene-Auto project!

13. References

[1] Toom, A., Naks, T., Pantel, M., Gandriau, M., Indrawati: Gene-Auto - an Automatic Code Generator for a safe subset of Simulink-Stateflow and Scicos. In: European Congress on Embedded Real-Time Software (ERTS), Toulouse, 29/01/2008-01/02/2008 http://www.sia.fr, 2008

[2] Izerrouken, N., Thirioux, X., Pantel, M., Strecker, M.: Certifying an Automated Code Generator Using Formal Tools : Preliminary Experiments in the GeneAuto Project. In: European Congress on Embedded Real-Time Software (ERTS), Toulouse, 29/01/2008-01/02/2008 http://www.sia.fr, 2008

[3] The EDONA project: "*Plate-forme EDONA Studio et référentiel commun*". http://www.edona.fr/scripts/home/publigen/content/templates/show.asp?P=124&L=FR&ITEMID=21

[4] The EDONA project: "*EDONA Newsletter n3 December 2009*". http://www.edona.fr/home/liblocal/docs/EDONA_Newsletter_n3_December_2009-v1.pdf, 2009

[5] Besnard, L., Gautier, T., Ouy, J., Talpin, J.-P., Bodeveix, J.-P., Cortier, A., Pantel, M., Strecker, M., Garcia, G., Rugina, A., Buisson, J., Dagnat, F. Polychronous Interpretation of Synoptic, a Domain Specific Modeling Language for Embedded Flight-Software. In Bujorianu, M. and Fisher, M. (editors), Workshop on Formal Methods for Aerospace (FMA) EPTCS 20, 2010

[6] Barnes, J. High Integrity Software: The SPARK Approach to Safety and Security. Addison-Wesley. ISBN 0-321-13616-0. http://www.praxis-his.com/sparkada/sparkbook.asp, 2006

[7] Izerrouken, N., Pantel, M., Thirioux, X., Machine-Checked Sequencer for Critical Embedded Code Generator, Springer LNCS proceedings of ICFEM'09, 09/12/2009-12/12/2009, Rio De Janiero – Brazil, 2009.

[8] Bruno Barras and Bruno Bernardo. The implicit calculus of constructions as a programming language with dependent types. In Roberto M. Amadio (editor), FoSSaCS, volume 4962 of Lecture Notes in Computer Science, pages 365–379. Springer, 2008.

[9] Pierre Letouzey. Extraction in coq: An overview. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, CiE, volume 5028 of Lecture Notes in Computer Science, pages 359–369. Springer, 2008.

[10] Stéphane Glondu. Extraction certifiée dans Coq-en-Coq. Journées Francophones des Langages Applicatifs (JFLA2009), 2009.

[11] 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.

[12] Pnueli, A., Siegel, M., Singerman, E.: Translation validation. In: Proceedings of the Tools and Algorithms for Construction and Analysis of Systems Conference (TACAS'98). Volume 1384, pages 151–166, 1998

[13] Necula, G.C.: Translation validator for an optimizing compiler. ACM SIGPLAN Notices 35(5) pages 83–94, 2000

[14] Leroy, X.: Formal certification of a compiler back-end or : Programming a compiler with a proof assistant. Proceedings of the 33rd Symposium on Principles Of Programming Languages (POPL'06) 41(1) pages 42-54, 2006

[15] Rugina, A.-E., Dalbin, J.-C.: Experiences with the GENE-AUTO Code Generator in the Aerospace Industry. In: European Congress on Embedded Real-Time Software and Systems ($ERTS^2$), Toulouse, 19/05/2010-21/05/2010 http://www.sia.fr, 2010

14. Glossary

MDE: Model-Driven Engineering

ACG: Automatic Code Generator

MOF: Meta Object Facility

EMF: Eclipse Modeling Framework

OCL: Object Constraint Language

ATL: ATLAS Transformation Language