Generative Programming for a Component-based Framework of Distributed Embedded Systems

Xu Ke, Krzysztof Sierszecki

Mads Clausen Institute for Product Innovation, University of Southern Denmark
Grundtvigs Alle 150, 6400 Soenderborg, Denmark
{xuke, ksi}@mci.sdu.dk

Abstract. COMDES-II is a component-based software framework which formally specifies the modeling concepts and constraints for distributed embedded systems in different aspects, such as component structures, interaction, hierarchy, etc. The paper presents an overview of the design philosophies of COMDES-II in the related aspects and a generative programming approach developed to enable the engineering applicability of the framework. The dedicated generative programming approach involves the formal definition of COMDES-II modeling language by means of meta-models which are instrumented by a meta-modeling tool – Generic Modeling Environment (GME), and the development of a specific code generation technique using CodeWorker tool to implement the automatic synthesis of system codes from system models.

1 Introduction

Complexity of software in embedded applications is continuously increasing, this situation is caused – primarily – by growing computational power of general-purpose microprocessors, which eliminates the need for special dedicated hardware solutions and therefore moves the emphasis from hardware to software design. An embedded software system is characterized by its tight interaction with the physical environment, restricted running resources (e.g. RAM, CPU, etc.), robust and strictly safe execution under hard real-time constraints [1]. These domain-specific features mandate the investigation of proper *domain-specific modeling* (DSM) techniques for various application domains of embedded systems, whereby the modeling concepts and abstraction rules of software that are provided in the solution space should accommodate the critical aspects in the problem space, such as system concurrency, environmental physicality, time, etc.

Component-based design (CBD) can be regarded as one of the most suitable design paradigms (if not the most suitable) for domain-specific modeling methodology. Due to the great profits brought by reusability of components, higher level of system abstraction (modeling systems rather than programming systems), an embedded software system can be efficiently and intuitively constructed from the prefabricated and reusable components. Moreover, from a software engineering point of view, CBD is an effective way to bridge the gap between the conceptual system design models and the concrete system implementation [2], provided that a proper *generative programming* approach is developed.

Generative programming is a software engineering methodology that automates the generation of system implementations from higher level abstractions represented as textual or graphical models [3]. In this context, *meta-modeling* and *model-driven development* (MDD) techniques provide great advantages for modeling domain-specific software systems at higher abstraction level, and on the other side, *code generation* and *model transformation* are the general approaches adopted to implement the automatic synthesis facilities for systems.

This paper intends to present such a generative programming method for a domain-specific, component-based software framework aiming at the development of distributed embedded systems – *COMDES-II* (Component-based Design of Distributed Embedded Systems, version II) [4]. The focus is placed on the design philosophy and the meta-modeling approach of framework components in the component design aspect, and the code generation technique related to the component implementation aspect (as shown in Fig. 1).

The meta-models of COMDES-II components are represented as the special UML class diagrams provided by the meta-modeling tool *GME* (Generic Modeling Environment) [5, 11], a configurable toolkit that supports the creation of domain-specific modeling and program synthesis environments. The constraint language *OCL* (Object Constraint Language, a subset of UML 2.0) is also supported in GME, which can be used to help specify the complex static semantics of component models in COMDES-II.

For the development of code generation technique, the *CodeWorker* [12] tool is employed. CodeWorker is a versatile parsing tool and a universal generator, which provides a scripting language adapted both to the description of any input format and to the writing of any generation templates [6]. COMDES-II models are parsed by an extended-BNF script to create a parse tree, which is subsequently processed by template-based scripts that drive the code generation. The generated code and the reusable component execution algorithms are finally composed into the executable code by means of the GNU Compiler Collection (GCC) [13].

This engineering approach involving the graphical modeling of COMDES-II components and the automatic synthesis of component codes can be conceptually illustrated as in Fig. 1.

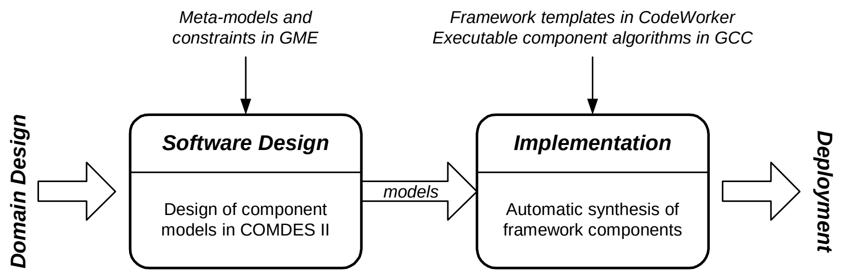
Meta-models and constraints in GME
Framework templates in CodeWorker
Executable component algorithms in GCC
Domain Design
Software Design
Design of component models in COMDES II
models
Implementation
Automatic synthesis of framework components
Deployment

Fig. 1. The generative programming approach for COMDES-II components

Firstly, a component is designed in its application domain, at relatively high level, e.g. a controller of control system in MATLAB/Simulink [14]. Next, the domain component model that satisfies application requirements can be transformed into the COMDES-II framework model, e.g. by automatic mapping from Simulink components to COMDES-II components. The transformed framework components may have some supplementary information which will guide the implementation generation. Ultimately, the synthesized code can be deployed into embedded devices and tested against real environment.

The paper is organized as follows: Section 2 gives a brief introduction about the design philosophies of COMDES-II. Section 3 presents the meta-level definitions of COMDES-II components through an example. Section 4 describes the code generation technique developed under CodeWorker tool to automatically synthesize the framework code from the corresponding component models, and the concluding section summarizes this engineering approach for COMDES-II framework, discusses the related research and future work.

2 The COMDES-II Framework

COMDES-II is a component-based framework with its focus on the distributed control systems, as a result the framework places its root in the control engineering domain and borrows a number of software concepts that are popular in this domain, such as function blocks, state machines, etc. [7].

COMDES-II provides specific modeling techniques in the solution space by emphasizing two significant aspects of an embedded software system: 1) the openness and hierarchy of system architecture, and 2) predictable and deterministic system behaviour, by taking the following problem space issues into account:

- *Component structures, interaction and hierarchy*
- *System architecture, concurrency*
- *Environmental physicality (e.g. external events etc.) and time*

The framework employs a two-level architectural model to specify the system architecture: at the first level (system level) an embedded application is conceived as a composition of *actors* (active components) that exchange signals asynchronously through a content-oriented, producer-consumer model of communication. An example of the system developed under COMDES-II for Production Cell Case Study [8] is shown in Fig. 2.

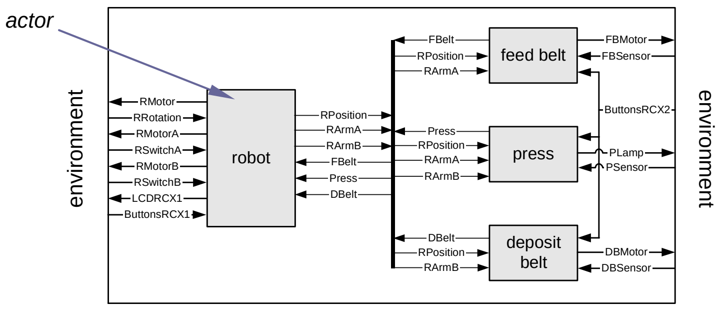

Fig. 2. Actors interaction in COMDES-II

At the second level (actor level), an actor contains multiple *I/O drivers* and a single *actor task* (execution thread). I/O drivers are classified as *communication drivers* and *physical drivers*, which are associated with the actor task by the *dataflow* connection relationship. As an example, the internal structure of feed belt actor shown in Fig. 2 is illustrated as in Fig 3.

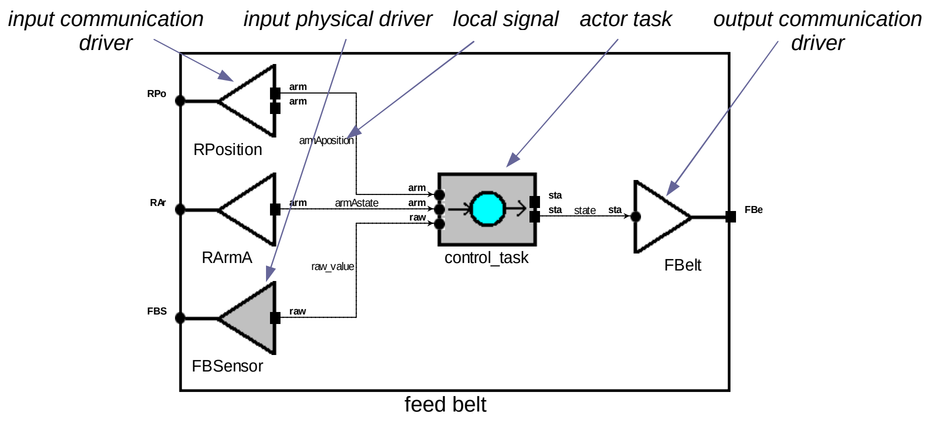

Fig. 3. Internal structure of the feed belt actor

The I/O drivers of an actor are assumed to be short pieces of code executed atomically (zero time) at precisely specified time instants referred to as *execution triggering* instant and *deadline* instant respectively, hence the execution triggering instant of an *actor* is also the *releasing* instant of the *actor task*. The actor tasks and I/O drivers are scheduled by the real-time kernel *HARTEX*$_{TM}$[1] [9], which employs a preemptive priority-based *timed multitasking* (TM) technique [10]. TM guarantees the execution time of an actor is constant – nevertheless the actor task may be preempted by higher priority tasks in arbitrary times – as long as the task finishes execution before its deadline. This execution pattern of actor tasks is referred to as *split-phase* execution and illustrated by the diagram shown in Fig. 4.

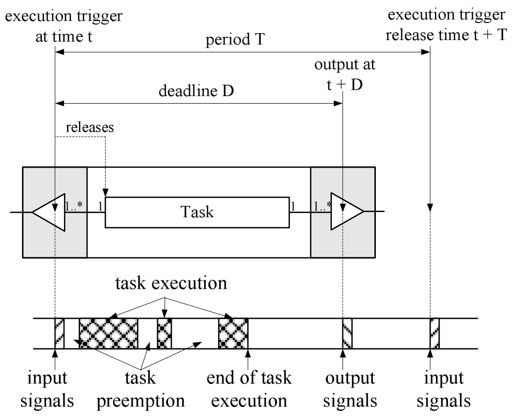

Fig. 4. Split-phase execution of actor tasks under timed multitasking

An actor task can be hierarchically composed from an aggregation of different *function block instances* (passive components). Function block (FB) instances are instantiations of reusable FB *types*, which can be categorized into four FB *kinds* (meta-types) - *basic*, *composite*, *modal* as well as *state machine* FBs. A basic FB contains attributes, operations and associations that are common to all kinds of FBs, such as inputs, outputs, parameters, etc. Hence the definition of basic FBs is a root class which can be extended to define the other kinds of FBs. A more detailed description of each kind of FBs is referred to [4]. And as an example, the FB instances contained in the feed belt actor *task* (named control_task) are shown as in Fig. 5.

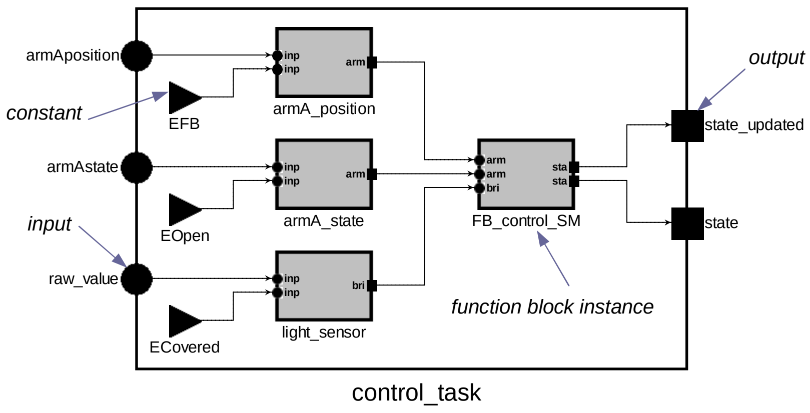

Fig. 5. Internal structure of the feed belt actor task

The concrete operation dynamics of this actor and its constituents will not be explained here since they are irrelevant to the focus of discussion, and we hope the diagram is intuitive enough to demonstrate the architectural and hierarchical features of COMDES-II framework.

[1] *HARTEX*$_{TM}$ is a hard real-time kernel developed by Software Engineering Group, Mads Clausen Institute for Product Innovation, University of Southern Denmark (SEG, MCI/SDU).

A FB type is a software component with an *execution record* containing its attributes and a set of *operations* defining its possible behaviour. A generic component model for all kinds of COMDES-II FBs is conceptually illustrated as in Fig. 6.

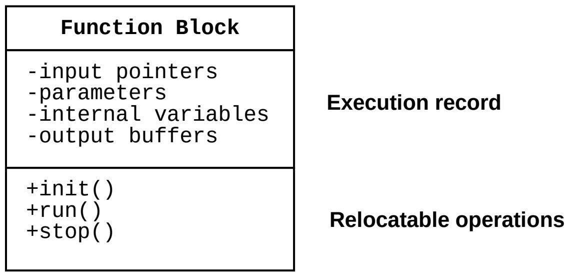

Fig. 6. Component model for COMDES-II FBs

The execution record is actually the FB interface containing the information like input pointers, parameters, internal variables and output buffers of a specific type of FB. FB execution record can be instantiated as well as reconfigured for the related FB instances of a given type. The operations are reentrant and relocatable functions performing some kinds of algorithms on an execution record, by accepting a pointer as the argument referring to the corresponding execution record of a specific FB instance.

In COMDES-II, the interface of a specific FB type can be automatically synthesized into the C files from the corresponding FB graphical design model. The operations of a given type of FB are predefined algorithms and implemented as C routines. The prefabricated operation and interface files of FB definitions are stored in the FB repository, in which the operation files are delivered as executable routines, e.g. as object files (.obj files).

3 Meta-level Definitions of COMDES-II Components

The description of COMDES-II framework presented in the previous sections is informal, which is helpful to intuitively understand this DSM framework though, it is yet insufficient to implement a DSM language that is compliant with the framework rules and constraints. A DSM language of COMDES-II enables the modeling of components and application systems under the framework, and in order to develop such a language, the meta-models formally describing the syntax and static semantics of the targeting domain modeling language should be defined with a consideration of various problem space issues (e.g. hierarchy, time etc.). Generally speaking, the formalization of modeling languages to be the corresponding meta-models is a recursive process which can be conceptually presented as in Fig. 7.

Meta-modeling COMDES-II framework involves the formal specification of following abstractions in different aspects:

- Meta-modeling *HARTEX*$_{TM}$ kernel and actors to accommodate the physicality (handling external interrupts), actor task concurrency (primitive priority-based scheduling), actor interaction (actor communication, actor synchronization etc.) and timing aspects (timed multitasking).
- Meta-modeling various kinds of function blocks in terms of function block structures, function block interaction and hierarchy (e.g. a model function block can contain other function block instances).
- Integrating the meta-model of *HARTEX*$_{TM}$ kernel and actors with the function block meta-models to accommodate the architectural aspect of the framework.

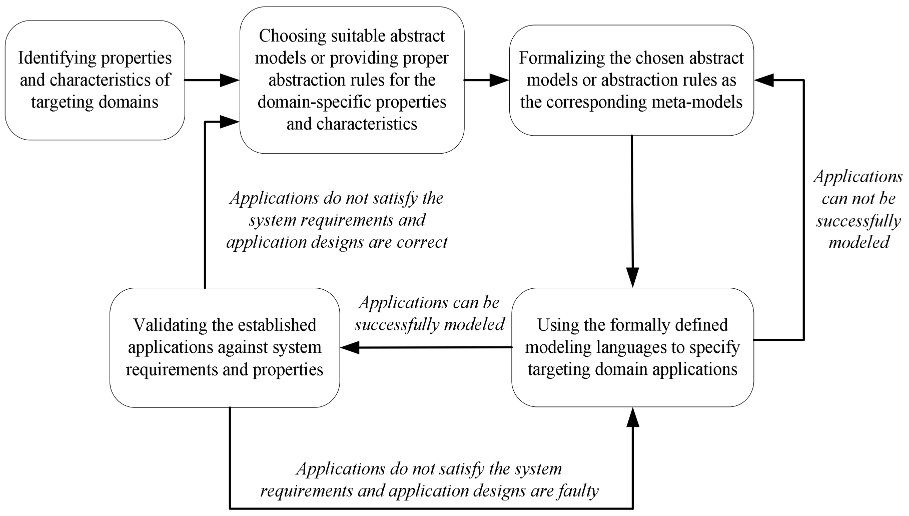
Identifying properties and characteristics of targeting domains
Choosing suitable abstract models or providing proper abstraction rules for the domain-specific properties and characteristics
Formalizing the chosen abstract models or abstraction rules as the corresponding meta-models
Applications can not be successfully modeled
Applications do not satisfy the system requirements and application designs are correct
Validating the established applications against system requirements and properties
Applications can be successfully modeled
Using the formally defined modeling languages to specify targeting domain applications
Applications do not satisfy the system requirements and application designs are faulty

Fig. 7. General meta-modeling process

In order to better understand the above meta-modeling approach, an example for formalizing the models of state machine FBs (SMFBs) is given. A SMFB in COMDES-II employs a dialect of the finite state machine model with event-driven semantics to specify the sequential behavior of a system. The graphical representation of FB_control_SM function block instance in Fig. 5 is presented as in Fig. 8.

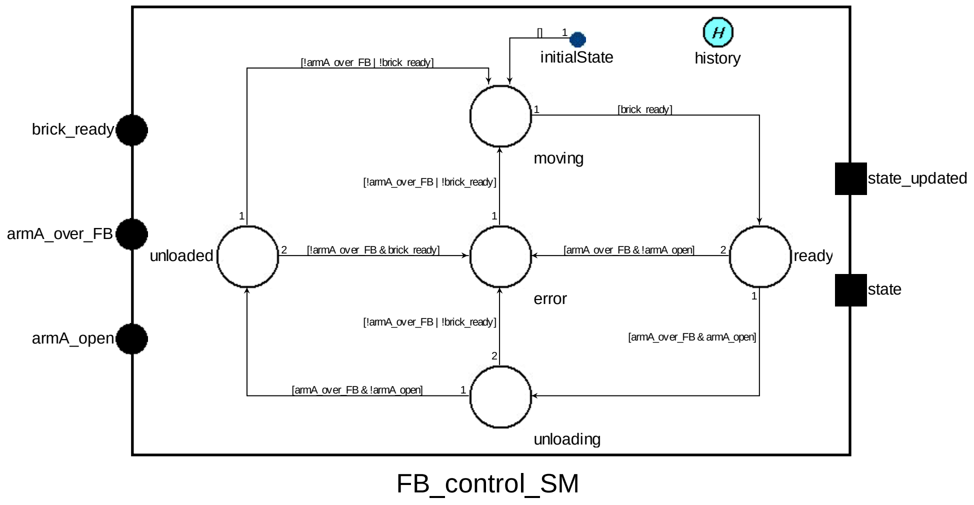
[] 1
initialState
history
[!armA_over_FB | !brick_ready]
1
[brick_ready]
moving
[!armA_over_FB | !brick_ready]
brick_ready
state_updated
armA_over_FB
1
1
unloaded
2
[!armA_over_FB & brick_ready]
[armA_over_FB & !armA_open]
2
ready
error
1
state
[!armA_over_FB | !brick_ready]
[armA_over_FB & armA_open]
armA_open
2
[armA_over_FB & !armA_open]
1
unloading
FB_control_SM

Fig. 8. FB_control_SM function block instance

This function block instance contains three inputs and two outputs, which are the common elements that all kinds of function blocks have and therefore are inherited from the basic function block definition. Additionally, an event-driven state machine model specifying the sequential behavior of the host actor is also integrated. The state machine model includes a dummy initial state pointing to the actual initial state of the machine, a graphical label with the name history meaning that the state machine is historic, a number of states as well as state transitions which are labeled by events, guards and transition orders. Transition order is a number indicating the importance of the transition, i.e. which transition should be fired when multiple transition triggers associated with the current state are evaluated as true (transitions are evaluated starting from 1).

The above informal abstractions of the state machine function block can be formalized by a meta-model as illustrated in Fig. 9.

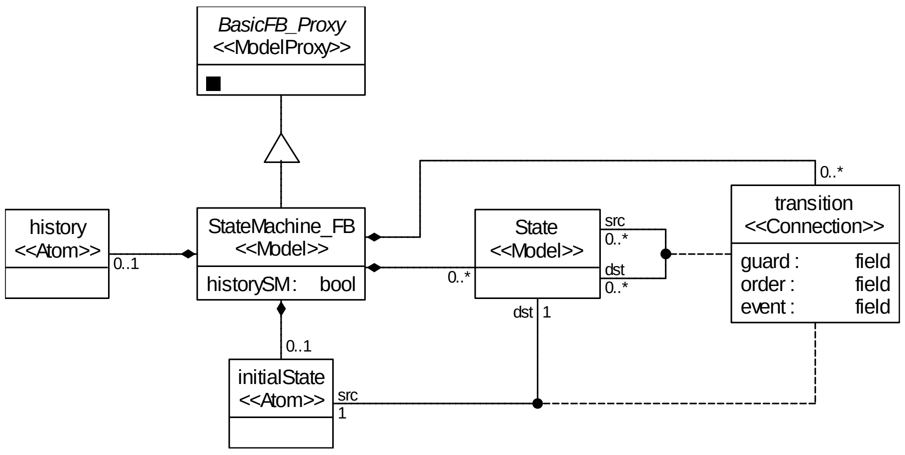

Fig. 9. Meta-model of state machine function blocks

In addition to the meta-model defined as a class diagram, some extra constraints specifying the static semantics for the state machine model are also defined in OCL, which are summarized as in Table.1. The meta-model in form of class diagram together with the constraints expressed in OCL provide a complete formal definition for the corresponding kind of function block.

Table 1. Example of constraints in OCL

Syntactic Constraint	OCL Expressions	Applying Object	Checking on Event
The state machine is reactive.	`self.models("State")->forAll(s | s.connectedFCOs("dst")->size >= 1)`	StateMachine_FB	CLOSE_MODEL
The state machine is deterministic.	`self.connectionPoint("src").target().attachingConnections("src","transition")->select(c : transition | c.event = self.event and c.guard = self.guard)->size = 1`	transition	CONNECT
All states are reachable	`self.models("State")->forAll(s | s.connectedFCOs("src")->size >= 1)`	StateMachine_FB	CLOSE_MODEL

4 Code Generation Technique of COMDES-II Framework

Implementation of COMDES-II system is achieved in two stages: firstly, CodeWorker generates source code files from GME models; secondly, GCC composes the generated source files with prefabricated codes into the final executable implementation. Execution of the first stage is controlled by an application written in Java accessing CodeWorker functionality via its Java interface, whereas the second stage is conducted by the Makefile generated in first stage.

COMDES-II implementation is built, or configured from predefined and reusable components stored in a repository. For each application component instance a data structure

(FB execution record) is generated, whereas the accompanying component algorithms (FB operations) are prefabricated in advance. In this way, during application synthesis no component executable code is generated.

In order to match the limited resources of embedded systems, COMDES-II framework is implemented in C language, which could be seen to some extent as a portable source code as long as the GCC tool chain is employed. Because some parts of the C code (e.g. FB operations) are only CPU architecture dependant and are compiled into an executable object codes for a particular CPU architectures, e.g. avr5 – ATmega128. Some parts, as usual, are dedicated to a particular hardware platform (e.g. hardware I/O drivers) and are written by an expert once (Fig. 10). In this way, portability and native platform performance is achieved rather easily, assuming existence of GCC tool chain for the platform of interest.

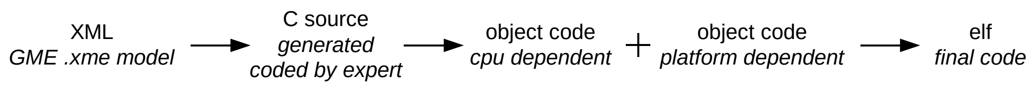

Fig. 10. Portability of COMDES-II system

An overview of the generation process is given in Fig. 11, with three different scenarios:

- Application synthesis (green, solid line) – models, which provide all necessary information, drive the configuration of an application.
- Component generation (blue, dashed line) – component execution record is generated, and then supplemented with the algorithms written by software expert. Final implementation is stored in a repository of reusable components in a form of executable object file.
- Reconfiguration (red, dotted line) – rather than generating the reconfigured application as a whole, only the updated part is created, which provides for faster application modification.

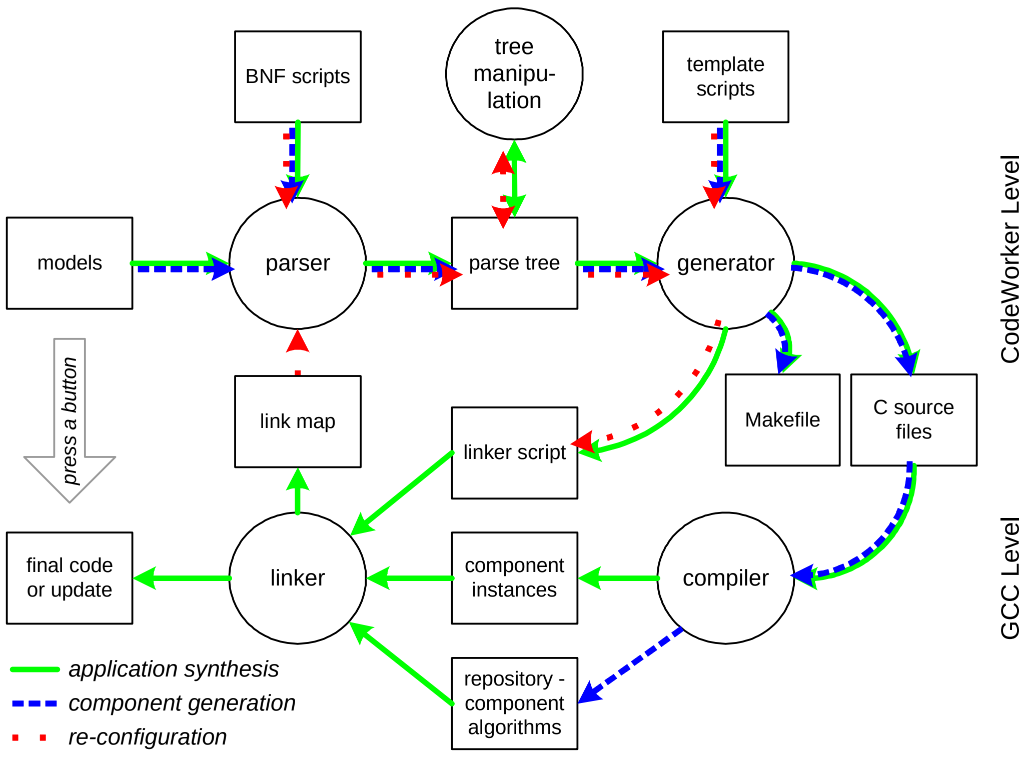

Fig. 11. Automatic synthesis of COMDES-II system

5 Conclusion

COMDES-II is a component-based framework aiming at the software development in the domain of distributed embedded systems. The framework provides the modeling methods for domain-specific features of an embedded system in different aspects, including component structures and interaction, system concurrency and functionality under the hard real-time constraints, etc. The provided design methods in these aspects enable COMDES-II a framework accommodating both the open system architecture as well as the predictable and deterministic system behaviour.

In the paper a generative programming approach for COMDES-II has been presented, which involves the meta-modeling of framework modeling language and the development a dedicated code generation technique. A complete formal definition of the COMDES-II components carried out in GME consists of a meta-model specified as a class diagram and a set of constraints expressed in OCL, which is exemplified in the paper with a concrete state machine function block instance. Automatic synthesis of application implementation is a process consisting of parsing of models and generating source files in CodeWorker, next, compiling and linking of all codes in GCC. Ultimate result is the configuration of applications from reusable and reconfigurable components.

Throughout the development of the generative approach we follow a motto: *let the best tool do the job, the tool that is designed for the job*. And therefore we adopt: GME – rapid development of DSM editor prototypes, CodeWorker – generation of any output and GCC – compiling and linking of codes. There are also other options of tools which can be used to develop the graphical DSM editor, for instance, Eclipse EMF/GMF/GEF frameworks [16], or MetaCASE MetaEdit+ [15].

Eclipse EMF/GMF/GEF frameworks provide an excellent model-driven approach for creating domain-specific models from their meta-models, and allow developers to establish a very flexible graphical environment for editing the models, however, developing such a graphical editor is really a labor-intensive task. MetaEdit+ is a commercial meta-modeling product developed by MetaCASE, which offers a Symbol Editor that facilitates the customization of model visual effects and a promising code generation tool for easy automatic synthesis and documentation. However, the meta-modeling process in MetaEdit+ is not as straightforward as that in GME or Eclipse EMF, and moreover, only the cardinality constraints of relationships are supported in MetaEdit+, whereas the Object Constraint Language (OCL) is not implemented. GME enables a powerful meta-modeling capability by providing a number of unique meta-modeling concepts, such as sets, references and aspects etc., additionally the OCL language is fully implemented. Automatic synthesis of program is also possible in GME through user-defined plug-ins and Builder Object Network (BON) API. A deficiency of GME is that the graphical representation of models can not be dynamically changed, due to its fixed Model-View-Controller architecture.

The presented software framework has been experimentally validated through two case studies: the Production Cell Case Study [16] and the Steam Boiler Control Specification Problem [17]. The envisioned future work includes the development of a graphical editing toolset in Eclipse, and the meta-model as well as model transformations from GME to the developed graphical environment. Such transformations can be realized by using dedicated model transformation languages, just like GReAT – *G*raph *Re*writing *A*nd *T*ransformation language – for model transformations in GME [19].

References

1. Lee, E. A.: Embedded Software. Advances in Computers, Vol.56. Academic Press, London (2002)
2. Reekie, J., and Lee, E. A.: Lightweight Component Models for Embedded Systems. Technical Memorandum UCB/ERL M02/30, University of California, Berkeley, CA 94720, USA, October (2002)
3. Czarnecki, K., and Eisenecker, U. W.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional. 1st edition, June (2000)
4. Angelov, C., Xu Ke, and Sierszecki, K.: A Component-Based Framework for Distributed Control Systems, to be presented to the 32nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2006), Cavtat-Dubrovnik, Croatia, August (2006)
5. Ledeczi, A., Maroti, M., and Bakay, A. et al.: The Generic Modeling Environment. Workshop on Intelligent Signal Processing, Budapest, Hungary, May (2001)
6. Lemaire, C.: CODEWORKER Parsing tool and Code generator. User’s guide & Reference manual, Release 4.2, May (2006)
7. Lewis, R.: Modeling Control Systems Using IEC 61499. Institution of Electrical Engineers, (2001)
8. Maraninchi, F., and Remond, Y.: Applying Formal Methods to Industrial Cases: the Language Approach (The Production-Cell and Mode-Automata). Proc. of the 5th International Workshop on Formal Methods for Industrial Critical Systems, Berlin (2000)
9. Angelov, C., Berthing, J., and Sierszecki, K.: A Jitter-Free Operational Environment for Dependable Embedded Systems. In A. Rettberg et al. (Eds.): From Specification to Embedded Systems Application. Springer, (2005) 277-288
10. Liu, J., and Lee, E.A.: Timed Multitasking for Real-Time Embedded Software. IEEE Control Systems Magazine: Advances in Software Enabled Control, Feb. 2003 65-75
11. GME: http://www.isis.vanderbilt.edu/projects/gme/
12. CodeWorker: a parsing tool and a source code generator: http://codeworker.free.fr/
13. GCC, the GNU Compiler Collection: http://gcc.gnu.org/
14. MATLAB and Simulink for Technical Computing: http://www.mathworks.com/
15. MetaCase - Domain-Specific Modeling with MetaEdit+: http://www.metacase.com/
16. The Eclipse Graphical Modeling Framework: http://www.eclipse.org/gmf/
17. F. Maraninchi and Y. Remond: Applying Formal Methods to Industrial Cases: the Language Approach (The Production-Cell and Mode-Automata). Proc. of the 5th International Workshop on Formal Methods for Industrial Critical Systems, Berlin, 2000
18. J.-R. Abrial: Steam Boiler Control Specification Problem. http://www.informatik.uni-kiel.de/˜procos/dag9523/dag9523.html
19. G. Karsai, A. Agrawal, F. Shi, J. Sprinkle: On the Use of Graph, Transformation in the Formal Specification of Model Interpreters. Journal of Universal Computer Science, Special issue on Formal Specification of CBS, 2003