Formal Development and Automatic Code Generation: Cardiac Pacemaker

Dominique Méry and Neeraj Kumar Singh

LORIA, Université Henri Poincaré Nancy 1
BP 239 54506 Vandœuvre-lès-Nancy
{mery,singhnne}@loria.fr

Abstract. Formal methods are very efficient techniques for formal verification of a specification and to find errors in early stage of the system development. In order to generate a high quality code from a formal specification particularly in the embedded system is highly indispensable and a de-facto standard in many industrial application domains, such as medical, avionics and automotive control. This paper presents automatic source code generation from the developed formal specifications of a cardiac pacemaker. Cardiac pacing system is a Grand Challenge in the area of Software Verification. This paper includes an architecture of automatic code generation tool, summary of a formal development of the cardiac pacemaker using refinement techniques in Event-B, code generation of the developed formal model into C, C++, Java and C# using code generation tool EB2ALL, and finally the code compilation and execution.

Key words: Cardiac Pacemaker, Formal methods, Verification, Validation, Code Generation

1 Introduction

Formal methods are indispensable techniques in the area of the highly critical systems like medical, avionics and automotive control, where verification and validation (V & V) are main techniques which are applicable at different stages of the development life cycle. In the development of an embedded system, it is important to determine that developed system meets specifications and output of the system is correct. In the embedded system community, formal method-based development implies verification and validation, and generated proof obligations ensure the proof of correctness of a system model. Automatic code generators associated with the formal development tools can generate a software code from a formal specification, thus enabling a model developer to generate a source code automatically without knowing the target language syntax.

Tony Hoare suggested Grand Challenges for Computing Research to integrate the research community to work together towards a common goal, agreed to be valuable and achievable by a team effort within a predicted timescale. Verification Grand Challenge is one of them. From the Verification Grand Challenges, many application areas were proposed by the Verified Software Initiative.
The pacemaker specification [2–4] has been proposed by the software quality research laboratory at McMaster University as a pilot project for the Verified Software Initiative [5]. The challenge is characterised by system aspects including hardware requirements and safety issues. Such a system demands high integrity to achieve the safety requirements.

Proof-based development methods [6] integrate formal proof techniques in the development of software systems. The main idea is to start with a very abstract model of the system. Details and design choices are introduced in an incremental way. The correctness between two levels is ensured by refinement proofs. The refinement supported by the Rodin [7] platform guarantees the preservation of safety properties. Thus, the behavior of the final system is preserved by an abstract model as well as in the correctly refined models. Refinement-based model development is very popular in many industrial application domains. It is considered as a de facto standard in development of the very complex systems such as in medical, avionics and automotive domains.

Main contribution of this paper is to generate a source code from the proved formal specification of a cardiac pacemaker [8]. A complete formal development of cardiac pacemaker in Event-B [7, 6] is given in our published paper [8]. A refinement based technique is used to model a cardiac pacemaker system using Rodin tool [7] and inbuilt Rodin-prover [7] helps to discharge generated proof obligations to verify the consistency and correctness of the developed system. We have developed a set of translation tools; EB2ALL [9, 10], that transforms Event-B formal specification into many languages (C, C++, Java and C#). We have used this tool to generate the source code directly from formal specifications. Nevertheless all approaches which support formal development from specification to code must manage several constraining requirements, particularly in the domain of embedded software where specific properties on the code are expected.

The outline of the remaining paper is as follows. Section 2 presents the related work. A brief outline of the cardiac pacemaker system introduces in Section 3. Section 4 gives brief outline of the formal development of a cardiac pacemaker. Section 5 presents basic tool description of the EB2ALL and code generation process of the pacemaker model is presented by Section 6. Section 7 concludes the paper with some lessons learned from this experience and some perspectives along with future works.

2 Related Work

Macedo et al. [5] have developed a distributed real-time model of a cardiac pacemaker using formal tool VDM [11], where they have modeled the subset of pacemaker functionalities. In other pacemaker case study, Manna et al. [12] have shown a simple pacemaker implementation. Gomes et al. [13] have presented a formal specification of a cardiac pacemaker using Z modeling language and they have modeled the sequential model similar to Macedo et al. work [5]. They have also presented the pacemaker case study by providing a means to execute the model using a translation of Z model into Perfect Developer [14]. They have
used the existing tool Perfect Developer [14] to generate an executable code of the Z model.

We have presented a detailed formalisation of one- and two electrode pacemaker in [8]. The model has been developed in an incremental way using refinement approach in the Event-B modelling language. We have also presented a certification approach for medical devices using formal techniques in [15] and use a cardiac pacemaker challenge as a case study. In this paper, we have presented an extension work of our pacemaker model [8], in form of source code generation from Event-B model into C, C++, Java and C#. We have developed a tool EB2ALL [9, 10] that allows to transform any Event-B model into many languages with some constraints. This tool has been used here to obtain verified code from the proved formal specification of a cardiac pacemaker [8].

3 Overview of Pacemaker

The human heart is wondrous in its ability to pump blood to the circulatory system continuously throughout a lifetime. The heart consists of four chambers: right atrial, right ventricle, left atrial and left ventricle, which contract and relax periodically. Atria form one unit and ventricles form another. The heart’s mechanical system (the pump) requires impulses from the electrical system. An electrical stimulus is generated by the sinus node (see Figure 1), which is a small mass of specialized tissue located in the right atrium of the heart.

This electrical stimulus travels down through the conduction pathways and causes the heart’s lower chambers to contract and pump out blood. The right and left atrial are stimulated first and contract for a short period of time before the right and left ventricles. Each contraction of the ventricles represents one heartbeat. The atria contract for a fraction of a second before the ventricles, so their blood empties into the ventricles before the ventricles contract.

The conventional pacemakers serve two major functions, namely pacing and sensing. The pacemaker actuator is pacing by the delivery of a short, intense electrical pulse into the heart. However the pacemaker sensor uses the same electrode to detect the intrinsic activity of the heart. So, the pacemaker function of pacing and sensing activities are dependent on the behavior of the heart. The sensing and pacing functions regulate the heart rhythm. The pacemaker system is a small electronic device that helps the heart to maintain the regular heart beat.

An artificial pacemaker is implanted to assist the heart in case of a arrhythmias condition to control the heart rate [16]. Arrhythmias are due to cardiac problems producing abnormal heart rhythms, which are characterized by an irregular cardiac rhythm, e.g. due to asynchrony of the cardiac chambers. The irregularity of the heartbeat, called bradycardia and tachycardia. The bradycardia
indicates that the heart rate falls below the expected level while in tachycardia indicates that the heart rate goes above the expected level of the heart rate. An artificial pacemaker can restore synchrony between the atria and ventricles. In an artificial pacemaker system, the firmware controls the hardware such that an adequate heart rate is maintained, which is necessary either because the heart’s natural pacemaker is insufficiently fast or slow or there is a block in the heart’s electrical conduction system \[17, 16\]. Beats per minute (bpm) is a basic unit to measure the rate of heart activity.

The electrodes of the pacemaker are attached to the right atrium and the right ventricle. A cardiac pacemaker has several operational modes that regulate the heart functioning. The specification document \[2\] describes all possible operating modes that are controlled by the different programmable parameters of the pacemaker. All the programmable parameters are related to the real-time and action-reaction constraints, that are used to regulate the heart rate.

4 Formal Development of the Pacemaker

In one- and two-electrode pacemaker \textit{pacing} and \textit{sensing} activities are defined abstractly using \textit{action-reaction} and \textit{time patterns} \[6\]. We apply the action-reaction and time patterns in modeling to synchronize the sensing and pacing stimulus functions of the pacemaker system in a continuous progressive time constraint. We present here only summary informations about each refinement of one- and two-electrode pacemakers and omit detailed formalization and proof details. Complete formal development of the cardiac pacemaker is available in \[8\].

The following outline is given about every refinement level to understand the basic formal notion of the cardiac pacemaker model.

\textbf{Abstract Model} : Specifies the \textit{pacing} and \textit{sensing} under real-time properties using \textit{action-reaction} and \textit{real-time} patterns for defining abstractly initial events like \textit{Pace\_ON}, \textit{Pace\_OFF}, \textit{Sense\_ON}, \textit{Sense\_OFF} and \textit{tic} events.

\textbf{Refinement 1} : This refinement introduces additional features for filtering the exact sensing value through the pacemaker’s sensor by introducing standard threshold constants for both atrial and ventricular chambers, and new events are introduced as refinement of \textit{skip} for capturing the sensors value from the single or both chambers. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of a heart and a pulse generator for delivering stimulation pulses to the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value and a safety margin.

\textbf{Refinement 2} : This refinement introduces a \textit{hysteresis} operating mode to prevent constant pacing. The \textit{hysteresis} is a programmed feature whereby the pacemaker paces at a faster rate than the sensing rate. For example, pacing at 80 pulses a minute with a hysteresis rate of 55 means that the pacemaker will be inhibited at all rates down to 55 beats per minute, having been activated at a rate below 55, the pacemaker then switches on and paces at 80 pulses a minute \[16\]. The main purpose of hysteresis is to allow the patient to have his
or her own underlying rhythm as much as possible. In this refinement only new variables are introduced for applying hysteresis operating modes.

**Refinement 3:** In the final refinement, we describe a rate adapting pacing technique of the cardiac pacemaker. Rate modulation term is used to describe the capacity of a pacing system to respond to physiologic needs by increasing and decreasing pacing rate. The rate modulation mode of the pacemaker can progressively pace faster than the lower rate, but no more than the upper sensor rate limit, when it determines that heart rate needs to increase. This typically occurs with exercise in patients that can not increase their own heart rate. The amount of rate increase is determined by the pacemaker on the basis of maximum exertion performed by the patient. This increased pacing rate is sometimes referred to as the sensor indicated rate. When exertion has stopped, the pacemaker will progressively decrease the paced rate down to the lower rate. Two new events are introduced as refinement of skip for increasing and decreasing the pacing rate using accelerometer.

### 5 EB2C Translation Tool

Our approach is to translate the Event-B formal specification of a cardiac pacemaker using EB2ALL tool [9, 10]. EB2ALL is a collection of several plug-ins (EB2C, EB2C++, EB2J and EB2C#). We have developed all these tools and these are freely available for download. EB2ALL provides a fully automatic code generation approach from Event-B formal specification into many languages.

In this section, we demonstrate only basic architecture of code generation tool EB2ALL, which is a collection of a set of plug-ins (EB2C, EB2C++, EB2J and EB2C#) and all these tools have common architecture. Figure-2 depicts the common architecture of the EB2ALL tool [9]. This tool has mainly four components: Pre-processing, Event-B to target language translator, code optimization and code verification. Due to limited space, we will not describe development process of code generation tool. Complete architecture of the tool is given in [10].

---

1. Download: http://eb2all.loria.fr/
This paper [10] also explains code generation process and correctness of generated code, which comply with formal specification. To verify the correctness of generated code, a code verification technique like meta-proof and software model checking tools are used to validate the generated code with respect to the Event-B formal model. Main objective to use code verification to test the generated codes with proper coverage criteria and to ensure its correctness [9].

Figure 3 represents a screen shot of developed translation tool; EB2C under Rodin environment [9]. This tool is developed as a plug-in under the Eclipse framework. After installation of the EB2C plug-in, menu Translator/EB2C and tool buttons on toolbar, will appear. To generate source code in 'C' language of any formal model, a user can click on menu Translator/EB2C or tool button, then a dialog box will appear (see Figure 3). This dialog box presents a list of active projects. A user can select any project for 'C' code generation. This tool generates 'C' code for all concrete models and a log file for a selected project. We have used this tool to generate 'C' source code of our developed cardiac pacemaker model [8] using Event-B.

6 Code Generation for A Cardiac Pacemaker using EB2C

We have presented a proof-based an incremental formal development of a cardiac pacemaker in [8]. Here, we present an automatic code generation from developed formal specification of a cardiac pacemaker. We have used EB2C tool to generate 'C' code from Event-B model. This tool has a technique of automatic support of safety assurance of a generated code. To achieve a verified source code of the cardiac pacemaker, we have done further refinement of concrete model of the cardiac pacemaker using a new context, which has some data ranges corresponding to the 'C' language. This refinement makes model deterministic and generates some proof obligations due to defining the fixed data range of all the constants and variables of the cardiac pacemaker model. The generated proof obligations are discharged by automatic as well as manual and all these proofs are necessary to verify the specification in order to guarantee the consistency and correctness of the system. We have discharged all the generated proof obligations before generating the source code and move to the next level of code translation methodology as to pass the concrete model for continuing translation process.

The code translation from Event-B formal specification into 'C' programming language using EB2C tool [9, 10] is straightforward. The main idea is to translate an Event-B model into 'C' code using last concrete model. EB2C tool generates 'C' files corresponding to the concrete models. A generated 'C' file using EB2C tool has a basic structure: a set of constants, variables and functions. A set of constants and variables are extracted from the context and machines sections of the Event-B model, respectively. Data type of a constant is defined as an axiom in Event-B model. Similarly data type of a variable is extracted from the invariant section of the model. A set of constants and variables are given as follows, which are excerpted from the translated 'C' codes of the cardiac pacemaker model.

```c
enum status {ON,OFF}; /* Enumerated definition */
```
const int FixedAV=90; /* Integer in range 70–300 */
const int LRL=60; /* Integer in range 30–175 */
const int ARP=200; /* Integer in range 50–175 */
const int URL=120; /* Integer in range 50–175 */
const int VBlank =50;
const int PVARP=150;

enum status PMActuatorV; /* Enumerated type variable */
enum status PMSensorV; /* Enumerated type variable */
unsigned long int ThrV; /* Integer in range undefined */
unsigned long int AVCount; /* Integer in range undefined */

BOOL AVCountSTATE;
unsigned long last_sp;
unsigned long sp;
unsigned long PaceInt;

All the events of Event-B are translated into equivalent 'C' functions. An event INITIALIZATION is a 'C' function, which initialize default values of all the variables. An event of Event-B model has fixed organization of the internal components: local variables, guards (pre-conditions) and actions. An event may contain some local variables. The global constants and variables are declared on the top of the 'C' source file, while local variables are declared within the function body. All events of a formal model is translated as a set of 'C' functions.

During the translation of the events, the guards are translated into equivalent to 'if' statement using logical conjunction, disjunction, implication and equivalence. Each guard represents into a separate 'if' statement like nested 'if' structure. All these guards represent a set of preconditions, which are required to satisfy for executing the action predicates. All action predicates of a formal model event are directly translatable equivalent into 'C' assignment expressions. EB2C tool is capable to analyse the syntax of Event-B guards and actions predicate. All pre-conditions or guards are required to be TRUE for firing all the set of actions.

If all guards are true, then the actions predicates execute and return TRUE for successful execution of this 'C' function. If any 'if' condition false the this 'C' function returns FALSE and actions part of the function does not execute.

BOOK Actuator_ON_V (void)
{
    /* Guards No. 1*/
    if (PMActuatorV == OFF)
    {
        /* Guards No. 2*/
        if ((sp == PaceInt) || ((sp < PaceInt) &&
                     (AVCount > VBlank) && (AVCount > FixedAV)))
        {
            /* Guards No. 3*/
            if (sp >= VBlank && (sp >= PVARP))
            {
                /* Actions */
                PMActuatorV = ON;
                last_sp = sp;
                return TRUE;
            }
        }
    }
    return FALSE;
}

To make the generated code executable, the EB2C tool generates an Iterate function that contains list of all functions in a calling order, which may be optimized or sequential. Another function is main(), which calls Iterate function. These two extra functions are used to make the generated code executable.

The source code is automatically generated in 'C' from the verified specification in less than five seconds. The generated code resulted in over 5000 lines in all...
the operating modes. Due to lack of space, we have presented a brief overview of the translation process from the Event-B specification of the cardiac pacemaker formal model into ‘C’ using EB2C tool. Based on this translation, we were able to automatically generate ‘C’ code and execute a simulation of the pacemaker. To find complete automatic generated ‘C’ codes of the cardiac specification using EB2C tool. Based on this translation process, we have generated ‘C’ code automatically and execute a simulation of the pacemaker. We have also generated source code into other languages (C++, Java and C#) from the cardiac pacemaker specifications [8] using EB2C++, EB2J and EB2C# tools with same nature of translation process.

7 Conclusion and Future Work

This paper presents our recent results in the area of formal development for automatic source code generation into many languages (C, C++, Java and C#) of the cardiac pacemaker, one of the challenges in software verification suggested by the Verified Software Initiative. This work is an extension of our previous work as a one step further for the cardiac pacemaker development. In the previous work [8], we have presented incremental proof-based formal development of the cardiac pacemaker using Event-B modeling language [6].

Main contribution of this paper is to describe an approach to translate formal model of a cardiac pacemaker into a higher level programming language like ‘C’, C++, Java and C#. This paper also introduces the main principles, rules and implementation solutions for translation tool to generate code [9, 10] from the Event-B formal specification. The syntax adopted is restrictive, but it already covers most numeric applications, supports powerful static analysis methods and generates fast and safe source code in programming languages. The translator provides useful assistance to human programmers by automatically adding comments, generating code for each process, optimizing expressions and partitioning event as well as data structures. Finally, we have shown a satisfactory result and demonstrate the ability to generate automatically source code from Event-B specification of the cardiac pacemaker in programming languages, which are comparable to a code written by hand with ordinary programming languages. The gains rely then on the guarantees provided by the use of a formal method and on the certification level which can be obtained by this way.

In order to assess overall of our approach, the complete development of the pacemaker has been presented to a group of pacemaker developers (French-Italian based pacemaker development company). The developers are satisfied by the result of pacemaker development using this refinement based incremental development and source code generation of verified formal models. So, we have considered that such kind of techniques for developing a complex system using formal methods and automatic code generation are very essential for industries, where safety and securities are main issues.

In future, we have plan to investigate some approaches to generate hardware prototypes from our automatic generated codes. Another interesting piece
of future work is to create some test scenarios to validate our system. Automatic verification of generated codes from formal model is most important and challenging task. The reason is that the preservation at the code level of the properties proved at the architectural level is guaranteed only if - the underlying platform is correct and correctness of final system when filling in the stubs for internal actions into the automatic generated code.

Acknowledgments. Work of Neeraj Kumar Singh is supported by grant awarded by the Ministry of University and Research.

References