The Cardiac Pacemaker: a Case Study of Medical Cyber-Physical Systems

---

Abstract— *Medical Cyber-Physical Systems (MCPS) are safety critical systems the failure of which can lead to harm humans, including loss of life. These systems require a verification and validation process from the early stages of their development, and must be certified to meet safety critical standards. In this paper, we examine different existing approaches in the development of an embedded real-time MCPS system (i.e., the cardiac pacemaker). We analyze the limits that the current methods present to develop healthcare systems, and we present current research in this area aimed at solving these limits taking into account the new challenges and research directions.*

Index Terms— Process control systems, Real-time and embedded systems, Design studies, Concurrent, distributed, and parallel languages, Pacemaker, Testing and Certification.

1 Introduction

The integration of physical processes with computation and communication has been called *Cyber-Physical Systems* (CPS) [1]. CPSs rely on sensing, processing and networking in order to provide a strong integration and coordination be-tween computing science, network communications and the physical world. CPSs sense and measure the physical environment and digitize it to be processed (i.e., the physical layer); the accurate characterization of statistical properties of these data workloads is a key point for CPS design. In [2], authors present a statistical physics-based approach to describe CPS workloads via fractal equations (i.e., platform layer), which allows the optimization of resource allocation, task mapping and scheduling to obtain an efficient, dependable, safe, reliable and secure a design (i.e., computational layer).

CPS technology improves quality of life, such as in personalized health care. The CPS target health care applications are called *Medical Cyber-Physical Systems* (MCPS). They are life-critical and context-aware and must also be adaptive, autonomous and functional in order to provide a high quality-of-life to the patient. In this new technology there are many challenges to be addressed, which require a new engineering system perspective, taking into account the application domain specific characteristics.

Several agencies, among them *The National Institute of Standards and Technology* (NIST), have studied this issue and have created a research agenda, which is described in a basic document [3], which studies and analyzes the main issues and challenges in the design, manufacture, certification, and use of MCPS. The challenges envisioned for the next 10 years include: *(i)* system integration, *(ii)* critical infrastructure, *(iii)* embedded real-time systems design, and *(iv)* validation and certification.

In this paper, we will discuss how these challenges in developing MCPS are addressed by several open research and development issues and we will present some existing work by focusing on the cardiac pacemaker as a case study. A pacemaker is a surgically implanted electronic device that regulates an erratic heartbeat (i.e., arrhythmia). It is most frequently used to speed the heartbeat of patients who have a heart rate well under 60 beats per minute (i.e., bradycardia case). In some cases, they are also used to slow a fast heart rate (i.e., tachycardia). Sensors and actuators are used to sense and pace the heart's atrium and ventricular chambers in restricted time.

The pacemaker presents stringent requirements, including highly precise timing characteristics, small memory footprints, flexible sensor and actuator interfaces, and robust safety characteristics. Moreover, the pacemaker is a safety-critical system with hard real-time requirements and functional correctness guarantees.

Note that as in others MCPS (e.g., chemotherapy administration or blood transfusion), the pacemaker involves complex collaborations among changing configurations of the human body, the software system and the hardware device (see Table 1). The physical layer (e.g.,

the desired behavior of the heart) is modeled with physical abstractions, while the computational layer that is usually an embedded system (e.g., the pacemaker controlled) is modeled using software abstractions that can be deployed using component-based technology.

The rest of the paper is set out as follows: Section 2 gives background information about the required pacemaker functionality and time requirements. Section 3 describes the use of high-level programming languages on the development of real-time embedded systems and existing solutions of the pacemaker implementation. Section 4 presents some existing examples about how verification and certification can be included in early design stages of the pacemaker development. Finally, Section 5 concludes this paper.

2 Background

The pacemaker needs to know about the intrinsic timing behaviour of the heart, since it uses this knowledge to monitor the time interval between natural heart beats to determine whether or not the heart should be placed [4]. In the normal sinus rhythm, electrical impulses have a rate of 60-100 times per minute.

2.1 The Heart Timing Requirements

The ECG shows a series of waves, which occur during each heart-beat and are printed on paper or displayed on a monitor (see Fig. 1). During each heartbeat, a series of waves occur in a normal record, which are named P, Q, R, S, and T and follow an alphabetical order. The P wave represents depolarization and contraction of the atrium (i.e., sensed atrial activity). The QRS complex represents the depolarization and contraction of the ventricles (i.e., sensed ventricle activity), and the T wave represents repolarization of the ventricles (i.e., ventricle recovery) because of expansion of the ventricle chamber. The number of waves may vary and other waves may also be present. The normal QRS duration for each beat is less than 0.10 seconds (i.e., 0.08-0.10); the Q-T interval is from 0.32 to less than 0.44 seconds for the male and less than 0.46 for the female. The P-R interval provides a value for the time taken for the

electrical impulse to travel from the atria to the ventricle. The normal P-R interval ranges from 0.12 to 0.22 seconds. Variation in any of these can lead to abnormal functioning of the heart.

2.2 The Cardiac Pacemaker Time Intervals

The pacemaker controls the heart rhythm through sensing and pacing operations [5]. In order to do this, the pacemaker controls five elements: *(i)* an atrium sensor, *(ii)* an atrium pulse generator, *(iii)* a ventricle sensor, *(iv)* a ventricle pulse generator, and *(v)* a rate modulation sensor. The *Atrio-Ventricular Interval* (AVI) is the time taken for ventricle fill after atrial contraction, which is less than 0.15 seconds. The *Ventriculo-Atrial Interval* (VAI) is the time between the ventricle activity and an atrial sense, which is less than 0.85 seconds. After sensing the ventricle activity, the pacemaker waits for the VAI sensing for an atrial activity.

If no activity is sensed during VAI, an atrial pace is generated. Similarly, if there is no ventricular activity after the AVI interval, a ventricle pace is generated. The paced AVI (typically 0.20 seconds) is longer than the sensing AVI (e.g., 0.170 seconds) since the pacemaker takes about 0.25-0.50 seconds to sense a spontaneous P wave. The pacemaker needs to ignore false atrial activity after a ventricle pace during 0.35 seconds; this is called the *Post-Ventricular Atrial Refractory Period* (PVARP) interval. Also, during the time of the T wave, the pacemaker must not produce any activity. This period is called the *Ventricular Refractory Period* (VRP), and it usually lasts 0.2-0.35 seconds. The PVARP may affect the upper tracking rate of the pacemaker, which is defined by the *Total Atrial Refractory Period* (TARP).

2.3 The Cardiac Pacemaker Modes

The code established in 2002 by the NASPE/BPEG, which consists of five letters (see Table 3) allows us to know in a simple way the mode and type of pacemaker. A DDDRV pacemaker, for example, refers to a pacemaker pacing both chambers (A+V), sensing both chambers (A+V), with the possibility to trigger and inhibit (I+T) depending on a biosensor (R), and stimulating both ventricles.

There are four different scenarios of the DDDR operating mode of the pacemaker: *(i)* normal behavior, *(ii)* atrial pace, *(iii)* ventricle pace, and *(iv)* both atrial and ventricle pace. Some modern pacemakers can have several operating modes (e.g., DDD and VDD), depending on the patient´s pathology, and can switch the operating mode automatically according to the patient's needs.

3 Embedded Real-Time System Design

Timely access and processing of sensed data from sensors are critical in MCPS. The elements to consider in designing real-time data processing for MCPS are data response time and data arrival updates. The pacemaker is a typical example of safety-critical systems and it is also a reactive system (i.e., waiting on the occurrence of input events, reacting to them by producing some outputs) with time constraints, which are the most important requirements and the most difficult implementation aspects. In this section, we provide an overview of how high-level programming languages extensions (see Table 4) have been used to implement a subset of the cardiac pacemaker specification.

3.1 The Pacemaker as a Real-Time System

The cardiac pacemaker is a real-time (reactive) system because its correct behavior depends on the logical order in which the events are performed as well as the timing of

the events. For example, after sensing activity in the auricle, the expected behavior is sensing the ventricle before AVI seconds.

Order: when activity is sensed in the auricle

Time: within AVI seconds

Note that the cardiac pacemaker is also a typical control program. Control programs are usually non-terminating, having the following logic program:

loop

read the sensor values at regular intervals

depending on the sensors' values,

trigger the relevant actuators

forever

During the time between consecutive readings of values of the sensors, the corresponding process enters a sleep state, and the CPU starts an idling mode to save power.

3.2 Real-Time Programming Languages

Ada 2005 has been widely adopted in safety-critical applications (e.g., the avionic and aeronautic industries). From 1998, the Java programming language has become an attractive choice for real-time systems because of its excellent software engineering characteristics. However, the standard Java is unsuitable for developing real-time embedded systems, mainly due to the under-specification of thread scheduling and the presence of garbage collection. These problems are addressed by the *Real-Time Specification for Java* (RTSJ).

Another Java-based technology specifically created to develop reactive systems is the *Globally Asynchronous Locally Synchronous* (GALS) SystemJ, which is a Java extension for reactive and concurrent systems. SystemJ includes both synchronous and

asynchronous concurrency, and allows the use of a framework based on a formal model of computation for the real-time specification, which provides correctness guarantees via temporal logic based verification.

In safety-critical systems, defined as systems the failure of which can lead to catastrophic damage, including loss of life (e.g., the cardiac pacemaker), time guarantees become essential and engineering is a complex task. The *Safety Critical Java* (SCJ) is a subset of RTSJ, created with the intention of certifying real-time Java applications using safety critical standards (e.g., DO-178B for airborne systems).

The *Reliable Ada Verifiable Executive Needed for Scheduling Critical Applications in Real-Time* (Ravenscar Ada) is the safety-critical subset of Ada 2005 for designing high integrity systems. Ravenscar Ada supports the time-triggered programming mode (i.e., as reactive programming languages). It has been created to accomplish two goals: a) deterministic and concurrent execution, suitable for hard (strict) real-time, and b) small and efficient implementation allowing tasks to meet tight deadlines with minimum overhead.

3.3 The Safety Critical Java Specification

In [6], the authors present a software pacemaker implementation in SCJ that has been structured in five handlers (i.e., two periodic handlers for atrial and ventricle sensor, two aperiodic handlers for the atrial and ventricle pulse generators, and a periodic handler for the rate modulation sensor). The periodic event handlers that are released at half of the P wave and the QRS complex monitor the sensors, record the times at which significant heart activities occur, and whether detect the absence of intrinsic heart activity program the corresponding timer for the aperiodic event.

Pacers are controlled by aperiodic activities. Whether a timer reaches 0, the corresponding event-triggered aperiodic handlers initiate the pacing. The event handlers

are scheduled by using a preemptive algorithm. Periodic event handlers are based on *Rate Monotonic Analysis* (RMA), which assigns the higher priority to the tasks with higher rate, and the aperiodic event handlers have the highest priority. Hence, the periodic handler for the rate modulation sensor has the low priority, the two periodic handlers for atrial and ventricle sensor have the same priority (i.e., a medium priority), and the two aperiodic handlers for atrial and ventricle pacing have the higher priority.

This solution can automatically switch from DDDR to DDIR mode and vice-versa. Hence, each mode has been encapsulated within a *mission*. The algorithm checks to see whether a mode change should occur. Since the pacing will occur before the mission is terminated, a spinning delay is required, waiting until the pacing end before the mission end.

3.4 Java-based Reactive Solutions

A SystemJ program may consist of one or more asynchronous concurrent behaviors (i.e., processes) called clock-domains. Each clock-domain is executed asynchronously, has its logical clock and can have one or more synchronous concurrent sub-behaviors (i.e., reactions), which execute concurrently advancing their states in lockstep. The pacemaker solution implemented in [7] uses a clock-domain (i.e., it uses only the synchronous subset of SystemJ). The process consists of three synchronous parallel reactions:

- The atrium reaction senses and paces the P wave from the atrium. If no P wave is detected within the VAI and after the PVARP timeouts, the atrium is paced.
- The ventricle reaction senses the QRS complex from the ventricle. If no activity is detected within the AVI time, the ventricle is paced.
- The operating mode handler (i.e., to change the operating mode from DDDR to DDIR and vice versa).

Reactions advance in lock step with the logical clock. There are three timers implemented as lower-level reactions. They run in parallel with the main control logic: *(i)* one timeout started after the last ventricular activity VAI, *(ii)* another after the last atrium activity AVI, and *(iii)* a third concurrent reaction implements the PVARP timeout, started after the last ventricular activity. All communication is via signals.

3.5 Ada-based Reactive Solutions

A Ranvenscar Ada solution with a similar structure to that given by SCJ is possible (i.e., sensors are aperiodic Ada tasks, and pacers are tasks waiting for their release event on an entry of a *protected object*). However, in [6] we found a more efficient solution, which requires less run-time support. This solution uses two timers: a watchdog timer, to detect the absence of intrinsic activities and to control the pacing current, and another periodic timer to initiate the reading of sensors. A single protected object encapsulates the handlers for these timing events, which avoids race conditions. All the activities (i.e., sensing and pacing) are controlled by timing events. As all code is running at interrupt level, it must be as simple and short as possible. Note that each handler must complete its execution before the other timing-events set in.

3.6 Comparison

The cardiac pacemaker is an interesting case study for both the real-time and the synchronous programming communities because it highlights the advantages and disadvantages of the classical scheduling model of real-time systems vs. the time-driven reactive model of synchronous systems, and also requires formal verification of functional and timing properties.

The SCJ implementation is simple and elegant, having 3 periodic tasks to sensors and 2 aperiodic tasks for pulse generators. The Ravenscar Ada solution has been implemented as a reactive system and it is similarly structured to that of the

SystemJ solution. Both, Ravenscar Ada and SystemJ, solutions have been implemented following the typical structure of synchronous programming language (e.g., Esterel). Whereas SCJ uses external timers combined with preemptive time-triggered handlers to implement the same functionality, and it is based on the classical real-time programming.

Table 4 shows how the solutions presented have been structured. Different than in SCJ, the clock driven execution of SystemJ implementation implicitly allows handling multiple events occurring simultaneously. This solution also uses formal semantics based on linear temporal logic providing verification of the time guarantees, which is normal in a reactive system. This is difficult to use in typical real-time systems.

The intrinsic heart activities are aperiodic and time-triggered in nature. Moreover, when considering an irregular ECG, pacing activities are not regular enough to be controlled by a periodic activity. So, the control requirements of the cardiac pacemaker are complex and do not conveniently fit the periodic programming paradigms supported by SCJ's and Ravenscar Ada's task model.

The Ravenscar Ada solution is more efficient than the SCJ-based one because handlers only execute when control is needed. However, since Ada 2005 requires that at least one task be present in the system to prevent program termination, it is necessary to introduce a dummy task in order to keep the program alive (i.e., Ada does not support the possibility of using only timing events to implement a reactive response), which can exhaust the pacemaker battery.

Regarding the SCJ-based solution, the spinning delay that maintains the current mission until the end of a pacing, whether a mode change occurs, wastes energy. Since the pacemaker is embedded within the human body, energy minimization becomes vital. Low energy consumption directly leads increasing device lifetime and reducing

maintenance costs. Moreover, with drastic increase in power density of modern electronic circuits, the temperature could affect system reliability and timing correctness. Hence, energy saving is crucial also due to the thermal issues.

3.6 Discussion

We consider the pacemaker as a real-time component (see Fig. 2), that is, as a complete computer system that is time-aware and consists of hardware, software, state (i.e., initial and history), and an interface with defined functionality and timing. The ideal implementation can along with the bests characteristics of the abovementioned solutions:

- A synchronous approach because its mathematical foundations provide formal concepts that favor the trusted design of embedded real-time systems. This model has several advantages: (*i)* it has formal and clear semantic definition, *(ii)* it reduces programming complexity, and *(iii)* many verification methods have been developed on the synchronous framework.
- A customized hardware-based implementation (e.g., a particular JVM hardware implementation executing a reduced bytecode subset) supporting the synchronous approach.
- Both, software and hardware must be verified and certified by the corresponding standards. The Ravenscar Ada profile and SCJ has been designed to accomplish with DO-178B certification, thus they include dead code elimination.

Here, the Java-based technology is the most adequate because we consider that it is possible to include the pure reactive programing in SCJ, and also to include the dead code elimination in systemJ. Along the last years, the pacemaker has been formalized using different tools (e.g., VDM, Z, Event-B, CPS, UPPAL). The systemJ solution that is based on the GALS model can benefit of this experiences.

The SPARK 2014 language consists of subset of the Ada 2012, which has been designed to be a formal method as defined by DO-333, and also supports both static and dynamic verification of contracts. A reactive SPARK-based implementation, where the waste of energy must be minimized, can be also a good solution. Also a hardware-based solution (e.g., FPGA) is also possible. However, software-based solutions allow us to address better complexity of the heart model allowing adaptive solutions.

4 CERTIFICATION AND VERIFICATION

The development of MCPS is a crucial issue for a country's economy. Thus, in [3] the main research areas we have identified procedures to ensure a safe healthcare have been identified, among them: *(i)* requirements and metrics for certifiable assurance and safety, *(ii)* formal methods, *(iii)* patient modeling and simulation, and *(iv)* adaptive patient-specific algorithms.

4.1 Assurance and Certification

Safety assurance and certification constitute a significant fraction of the development costs tasks in the development of safety critical systems, and is consequently a lengthy and expensive process. The pacemaker is faced with fairly sophisticated and complex risks with a reasonable probability of disaster (e.g., the death of the patient) if things go wrong. Because of this, it is important to use hazards analysis methodologies.

The main goal of safety analysis is to identify, eliminate, and control hazards. The ISO 14971 defines medical device risk-management, which includes policy development, training, and techniques such as *Failure Mode*, *Effect Analysis* (FMEA) and *Fault-Tree Analysis* (FTA) for determining the causes of system failures. The hazards analysis can become quite sophisticated and go into much detail. However, whether the potential hazards are significant and the possibility for trouble is quite real, such as in the pacemaker, this detail becomes essential.

The aim of assurance is to verify that a system enforces a desired set of security goals. In [9], the authors develop an assurance-case for pacemaker verification. The system assurance describes both what determines reasonable goal and what is a satisficing implementation, and also how the pacemaker should be built and maintained. This solution takes the requirement specified in [5] and focuses on time-guarantees and real-time software.

MCPS are also safety-critical and use standard protocols for assurance and certification. Concerning the pacemaker, the *Medical Device Directive* (MDD) 90/385/EEC establishes the guidelines for active implantable medical devices, and the IEC 60601 specifies requirements for electrical medical devices. The software must follow the IEC 62304 (i.e., the functional safety standard for medical device) class C. This standard specializes the IEC 61508-3 (i.e., general functional safety standard electrical/electronic programmable systems) for medical systems and presents three classes of safety criticality: *(i)* no injury or damage to health is possible for class A, *(ii)* no serious damage to health is possible for class B, and *(iii)* death or serious damage to health is possible for class C.

4.2 Formal Methods

Model-Based Development (MBD) has been playing a key role in the aeronautic and automotive industry for a long time, has enabled many advances, and is likely to hold the key to addressing many current and future challenges in MCPS. The design of MCPS with real-time and other critical constraints (e.g., space and power), as is the case of the pacemaker, raises specific problems throughout the development process. MBD provides means to capture architectural and non-functional aspects to separate functional aspects (i.e., platform independent).

4.3 Adaptive Patient-specific Algorithms

MCPS are typically designed for groups of patients with similar medical conditions. However, it is necessary to improve healthcare by adapting these systems to a specific patient with a specific medical condition. Machine-learning algorithms allow us to develop adaptive MCPS, which can be certified as safe for large classes of patients while adapting both to individual patients and to different environments.

The complexity of adapting the software pacemaker during runtime has spawned interest in how we can use MBD to manage and monitor its execution and extend the use of modeling techniques beyond the design and implementation phases (see Fig. 3). This adaptation requires research ideas and proposals from relevant areas such as software architectures as well as reflection and an autonomic and self-adaptive approach. In [11], we find an adaptive algorithm for the rate pacing, which is based on a predictive model that iteratively solves the optimal control with fractal state equations. The algorithm has been implemented on a FPGA and consists of four blocks: *(i)* observation, *(ii)* computation, *(iii)* control, and *(iv)* actuation.

4.4 Patient Modeling and Simulation

The pacemaker must adapt to the patient by taking into account different physical and environmental conditions. This requires developing models and simulators for model and virtual validation and testing. *Model Checking* (MC) is a finite state based technique for automatically verifying correctness properties, allowing find erroneous sequences of events and system states.

Given a model of a system, MC exhaustively and automatically checks whether this model meets a given specification. The specification can contain safety requirements such as the absence of deadlocks and critical states that can cause the system to crash. In [10], we find a *Virtual Heart Model* (VHM) for interactive test

generation, allowing the testing and verification of a pacemaker model in closed-loop. This solution models the heart timing and electrical impulses via a network of timed-automata, and has been mapped into two types of Simulink-based designs: a) based on a set of counters for periodically generated clock events, and b) based on absolute-time temporal logic to define time periods.

The testing of cardiac pacemakers poses numerous challenges because the interactions between the physical components and the cyber processes. The work presented in [13] describes a framework for validating time-bounded safety properties that is based on discrepancy functions. as well as determining the safe ranges for the system parameters. Timeline testing, which is based on finding the appropriate tests to provoke the system to miss a deadline, is the main challenge for the design and verification of cardiac pacemakers. Table 5 shows the categorization of testing techniques used in the studied solutions.

5 Conclusion

In this paper we have described and analyzed some relevant existing implementations of the cardiac pacemaker. The pacing of the ventricle/atrium is activated by events with periods that vary continually and even dramatically depending on the patient medical/physical conditions. This fact makes classical real-time techniques (e.g., RMA) inadequate due to its pessimism, which has direct implications on the life of the pacemaker battery. Alternatively, solutions based on reactive systems are aperiodic event-driven. However, these solutions are based on programmed timers, which make it difficult to take into account the range of possible heart acceleration and deceleration rates. Closed-loop, model-based solutions that take into account the interaction with the environment, can address the adequate pacing rate, which can improve the quality-of-life of the patient and the battery lifetime.

REFERENCES

[1] E.A. Lee, “CPS Foundations” In Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC), 2010. ACM, 737–742.

[2] P. Bogdan and R Marculescu, “Towards a Science of Cyber-Physical Systems Design”. IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS), 2011.

[3] I. Lee, G Pappas, “Report on the High-Confidence Medical-Device Software and Systems (HCMDSS) Workshop”, 2006, http://rtg.cis.upenn.edu/hcmdss/HCMDSS-final-report-060206.pdf

[4] D. L. Hayes and P. A. Levine , “Pacemaker Timing Cycles” In Cardiac Pacing and ICDs, 6th Edition, K.A. Ellenbogen and K. Kaszala Editors, May 2014.

[5] Boston Scientific, “Pacemaker System SpecificationTechnical Report”, 2007, http://www.cas.mcmaster.ca/sqrl/SQRLDocuments/PACEMAKER.pdf.

[6] N. K. Singh, A.J. Wellings, A. Cavalcanti, “The cardiac pacemaker case study and its implementation in safety-critical Java and Ravenscar Ada”. JTRES 2012.

[7] H. Park, A. Malik, M. Nadeem, Z. A. Salcic, “The Cardiac Pacemaker: SystemJ versus Safety Critical Java”, JTRES, 2014.

[8] D. Méry, N. K. Singh, “Closed-Loop Modeling of Cardiac Pacemaker and Heart”. FHIES 2012: 151-166

[9] E. Jee, I. Lee, and O. Sokolsky, “Assurance Cases in Model-Driven Development of the Pacemaker Software”. ISoLA 2, volume 6416 of Lecture Notes in Computer Science, page 343-356. Springer, (2010).

[10] Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam, "Real-Time Heart Model for Implantable Cardiac Device Validation and Verification", 24th Euromicro Conference on Real-Time Systems 2010, pp. 239-248.

[11] Z. Jiang and R. Mangharam, “Modeling Cardiac Pacemaker Malfunctions With the Virtual Heart Model”, 2011, http://repository.upenn.edu

[12] P. Bogdan, S. Jain, Siddharth, K Goyal, and R. Marculescu, “Implantable Pacemakers Control and Optimization via Fractional Calculus Approaches: A Cyber-Physical Systems Perspective”. IEEE/ACM Third International Conference on Cyber-Physical Systems, (ICCPS), 2012

[13] Z. Huang, C. Fan, S. Mitra, A. Mereacre, and M. Kwiatkowska, “Simulation-Based Verification of Cardiac Pacemakers with Guaranteed Coverage”, IEEE Design and Test, 2015, pages 27-34.