# The Cardiac Pacemaker Case Study and its implementation in Safety-Critical Java and Ravenscar Ada

Neeraj Kumar Singh
University of York, UK
neeraj.singh@york.ac.uk

Andy Wellings
University of York, UK
andy.wellings@york.ac.uk

Ana Cavalcanti
University of York, UK
ana.cavalcanti@york.ac.uk

## Categories and Subject Descriptors

D1.3 [**Concurrent programming**]: safety critical

## Keywords

safety critical programming languages, SCJ, Ravenscar Ada

## ABSTRACT

The cardiac pacemaker has emerged as a case study for evaluating the effectiveness of techniques for the verification and design of embedded systems with complex control requirements. This paper reports on the experiences of using this case study to evaluate the concurrency model of two programming language subsets that target safety-critical systems development: Safety-Critical Java (SCJ), a subset of the Real-Time Specification for Java, and Ravenscar Ada, a subset of the real-time support provided by Ada 2005. Our conclusions are that for SCJ, the lack of explicit support for watch-dog timers results in a software architecture where the time at which significant events occur must be saved, and polling must be used to detect their absence. Although this results in a less efficient system, the scheduling implications for the resulting software architecture are clear. In contrast, Ravenscar Ada's support for primitive timing events allow the construction of a highly optimized reactive solution. However, the timing properties of this solution are a little more complex to determine. Furthermore, the Ada solution requires a redundant task in order to prevent the program from terminating prematurely.

## 1. INTRODUCTION

Over the last decade, several Grand Challenges for Computing Research [12] have been set up as a means to encourage the research community to work together towards common goals that are agreed to be valuable and achievable by a team effort within a predicted timescale. One such challenge is the Verification Grand Challenge [11] whose goal is to improve the state of the art in software verification. The pacemaker specification [22] has been proposed as a pilot project for this challenge [25, 18]. It requires all system aspects, including hardware requirements and safety issues, to be considered.

The cardiac pacemaker is also a good example for evaluating the expressive power of a high-integrity real-time programming language, as it is a complex embedded real-time system. A pacemaker controls the heart rhythm through sensing and pacing operations; it has several operating modes, which are selected by the doctors. Some smart pacemakers automatically switch from one operating mode into another according to the patient's needs. Sensors and actuators are the two main components, which are used to sense and pace during restricted time intervals in both the heart's atrium and ventricular chambers. These sensors and actuators have time constraints, which are the most important requirements and the most difficult aspect to implement. The pacemaker is essentially a reactive system and its complexity comes from having to respond to the *absence* of intrinsic heart activity.

In this paper, we present our development of the software to support several complex operating modes using Safety-Critical Java (SCJ) [17] – the proposed safety-critical subset of the Real-Time Specification for Java (RTSJ)[5]. Our goal is both to evaluate the expressive power of the language and to provide an additional case study for those interested in programming safety critical systems (see [15, 9, 1] for related SCJ case studies).

Although the software is complex in its control requirements, it has limited demand for dynamic data; consequently we are evaluating the concurrency and timing models supported by SCJ, rather than its support for dynamic memory management. For the purposes of comparison, we also consider how the same software can be implemented in Ravenscar Ada [4] (the safety-critical subset of the tasking model of Ada 2005). This is a language that has been widely adopted in safety-critical applications.

Much of the literature related to the cardiac pacemaker development focuses on the formal semantics to verify the system properties. Macedo et al. [18] have developed a distributed real-time model of a cardiac pacemaker using the formal notation VDM. They have modeled a subset of the pacemaker functionalities. Tuan et al. [23] have proposed a formal model of the pacemaker based on its behaviour including the communication with the external environment. They have designed a real-time model using timed extensions of CSP and used the model checker Process Analysis Toolkit (PAT) in order to verify critical properties, such as deadlock freeness and heart rate limits.

In another study, Manna et al. [19] have shown a simple pacemaker implementation. Gomes et al. [8] have presented a formal specification of a cardiac pacemaker using the Z modeling language; they have covered the sequential model (given in Macedo et al. [18]) and provided a means to execute the model using a translation of the Z model into Perfect Developer. A detailed formali-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*JTRES '12, October 24 - 26 2012, Copenhagen, Denmark*
Copyright 2012 ACM 978-1-4503-1688-0 ...$15.00.

sation of the electrode pacemaker is presented by Mery et al. [21]. The model has been developed in an incremental way using refinements in the Event-B modelling language. They have also produced the code in various languages [20] from the validated formal specification of the cardiac pacemaker.

In all of the above approaches, any resulting programs (when given) consist of mainly code fragments with little consideration of how they are incorporated into a software architecture or how the components are scheduled to meet the required timing constraints. To our knowledge there has been no previous implementation of the pacemaker software in SCJ or Ada that considers the full software architecture.

It is beyond the scope of this paper to present an introduction to Safety-Critical Java or the Ravenscar subset of Ada 2005. The initial draft of the SCJ specification can be found in [17]. SCJ supports the notion of missions; a good introduction to this is given by Hunt and Nilsen [13]. The concurrency model is based on event handlers – see Wellings and Kim [24] for a rationale for this approach. Ravenscar Ada is a highly restricted subset of the full Ada concurrency and real-time models; see Burns [4] for the initial specification, and Burns and Wellings [3] for full consideration.

This paper is organized as follows. Section 2 presents a basic overview of the cardiac pacemaker and discusses its complex control requirements along with the required time constraints. Section 3 then presents the design and implementation of the SCJ software. SCJ does not provide full support for reactive system development, and this constrains the type of programs that can be designed. Ravenscar Ada, in contrast, does support the full time-triggered programming model. This leads to a different style for our Ada solution, presented in Section 4. Finally, Section 5 presents conclusions and future work.

## 2. CARDIAC PACEMAKERS

A pacemaker system is a small electronic device that helps the heart to maintain a regular beat. In this study, the pacemaker is treated as an embedded system operating in an environment containing the heart. The conventional pacemakers serve two major functions, namely *pacing* and *sensing*. The pacemaker's actuator paces 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's pacing and sensing activities are dependent on the behavior of the heart. We first review the heart system that interacts with the pacemaker (Section 2.1) and then consider elements of the pacemaker system itself (Section 2.2) along with its operating modes (Section 2.3). We conclude this section by discussing the control requirements.

### 2.1 The Heart System

The heart consists of four chambers (see Fig. 1): right atrial, right ventricle, left atrial and left ventricle, which contract and relax periodically. The heart's mechanical system (the pump) requires at the very least impulses from the electrical system. An electrical stimulus is generated by the sinus node, 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 chambers to contract and pump out blood. 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.

An artificial pacemaker is implanted to assist the heart in case of an arrhythmia condition in order to control the heart rate. Arrhythmias are due to cardiac problems producing abnormal heart

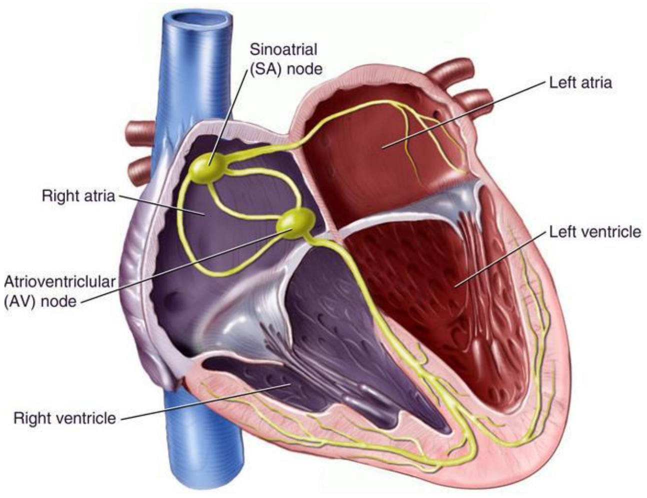


**Figure 1: Heart or Natural Pacemaker** (taken from http://media.summitmedicalgroup.com/media/db/relayhealth-images/nodes.jpg)

rhythms. An irregularity of the heartbeat can be bradycardia or tachycardia. Bradycardia indicates that the heart rate falls below the expected level, while tachycardia occurs when the heart rate goes above the expected level of the heart rate. An artificial pacemaker can restore synchrony between the atria and ventricles and can be used to treat bradycardia [2, 10].

| **Chambers Paced** | **Chambers Sensed** | **Response to Sensing** | **Rate Modulation** |
|---|---|---|---|
| **O**-None<br>**A**-Atrium<br>**V**-Ventricle<br>**D**-Dual(A+V) | **O**-None<br>**A**-Atrium<br>**V**-Ventricle<br>**D**-Dual(A+V) | **O**-None<br>**T**-Triggered<br>**I**-Inhibited<br>**D**-Dual(T+I) | **R**-Rate **Modulation** |

**Table 1: Operating Modes**

## 2.2 The Pacemaker System

The basic elements of the pacemaker system [2, 6] are:

1. **Leads:** One or more flexible coiled metal wires, normally two, that transmit electrical signals between the heart and the pacemaker. The same lead incorporate sensors, which are able to detect the intrinsic heart activity.

2. **The Pacemaker Generator:** This is both the power source and the brain of the artificial pacing and sensing systems. It contains an implanted battery and a controller.

3. **Device Controller-Monitor (DCM) or Programmer:** An external unit that interacts with the pacemaker device using a wireless connection. It consists of a hardware platform and the pacemaker application software.

4. **Accelerometer (Rate Modulation Sensor):** An electromechanical device inside the pacemaker that measures the body motion and acceleration of a body in order to allow modulated pacing. In the rate adaptive mode, a cardiac pacemaker automatically calculates the desire rate of the heart through the physical activities of the patient[16]. The rate modulation sensor is used to capture these physical activities and adjust the timing requirements for pacing.

The specification document [22] of our case study 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.

Fig 2 depicts a basic block diagram of the cardiac pacemaker and shows the sensors and actuators that will be monitored and controlled in the design presented in the remainder of this paper.

## 2.3 Operating Modes

In order to understand the pacemaker specification [22], it is necessary to comprehend the coding system that is used to describe a pacemaker's activities and requirements [28] (see Table-1). This is a code of five letters of which the first four are most often used. The code provides a description of the pacemaker pacing and sensing functions. The first letter of the code indicates which chambers are being paced; the second letter indicates which chambers are being sensed; the third letter of the code indicates the response to sensing; and the final letter, which is optional, indicates the presence of rate modulation in response to the physical activity measured by the accelerometer. "X" is a wildcard used to denote any letter (i.e. "O", "A", "V" or "D"). $Triggered$ $(T)$ refers to delivery of a pacing stimulus and $Inhibited$ $(I)$ refers to an inhibition from further pacing after sensing of an intrinsic activity from the heart chambers. Hence a mode **DDDR**, for example, indicates that the pacemaker is pacing in both (**D**ual) chambers, sensing in both (**D**ual) chambers, responding by triggering and inhibition (**D**ual) and modulates its activity according to physical activity of the patient (**R**ate).

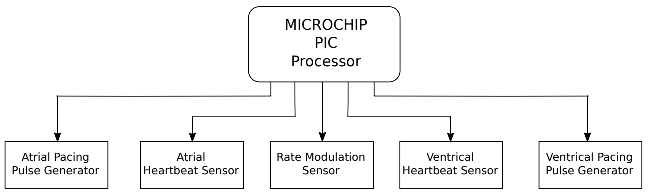

MICROCHIP PIC Processor
Atrial Pacing Pulse Generator
Atrial Heartbeat Sensor
Rate Modulation Sensor
Ventrical Heartbeat Sensor
Ventrical Pacing Pulse Generator

**Figure 2: Cardiac Pacemaker Sensors and Actuators**

## 2.4 Pacemaker Control Requirements

In this paper, we report our experience of developing an automatic mode switching algorithm of the cardiac pacemaker. Particularly, we consider two of the more complex operating modes (DDDR and DDIR) for automatic switching. The basic algorithm of both operating modes are the same. Therefore, this section presents only the control requirement of the DDDR operating mode. As explained above, the DDDR mode is a rate adaptive operating mode of the pacemaker, where the sensors sense intrinsic activities from both chambers and the actuators discharge electrical pulse in both chambers.

Fig. 3 depicts the scenarios for sensing and pacing activities [2]. The Ventriculoatrial interval (VAI) is the maximum time the pacemaker should wait after sensing ventricle activity (either intrinsic or paced) for some indication of intrinsic activity in the atrium. If none is present, the pacemaker should pace in the atrial chamber. The Atrioventricular Interval (AVI) is the maximum time the pacemaker should wait after sensing atrial activity (either intrinsic or paced) for some indication of intrinsic activity in the ventricles. If none is present then the pacemaker should pace in the ventricle chamber. After every pace in the ventricle, there is some sensed activity in the atrial, but this is not true intrinsic heart activity and should be ignored. The Postventricular atrial refractory period (PVARP) indicates the length of time that such activity should be ignored. Sensed atrial activity is called a P wave, and sensed ventricular activity is called a QRS complex. A T wave follows a QRS complex and represents the recovery of the ventricles. A P wave is sensed when the amplitude of the signal is greater than a threshold $P_{th}$ for $T_p$ time units. Similarly, a QRS complex is detected when the amplitude of the signal is greater than a threshold $QRS_{th}$ for $T_{QRS}$ time units. For safety, it is imperative to ensure that pulsing does not occur during a T wave.

There are four possible scenarios for pacing and sensing activities, which are given in Fig. 3.

- Scenario A – shows a situation in which the pacemaker paces after a standard time interval in both chambers. This is the reaction when no intrinsic heart activity is detected.

- Scenario B – shows a situation in which the pacemaker paces in the atrial chamber after a standard interval, while the ventricular pacing is inhibited due to a sensing of intrinsic activity from the ventricle.

- Scenario C – shows a situation in which intrinsic atria activity is sensed, pacing inhibited in the atrial chamber but occurs in the ventricular chamber after AVI (due to a lack of intrinsic ventricular activity).

- Scenario D – represents the case where both pacing activities are inhibited due to a sensing of intrinsic activities in both chambers.

The DDIR mode differs from DDDR only in the case of scenario C. In this situation, the intrinsic atrial activity occurs before the

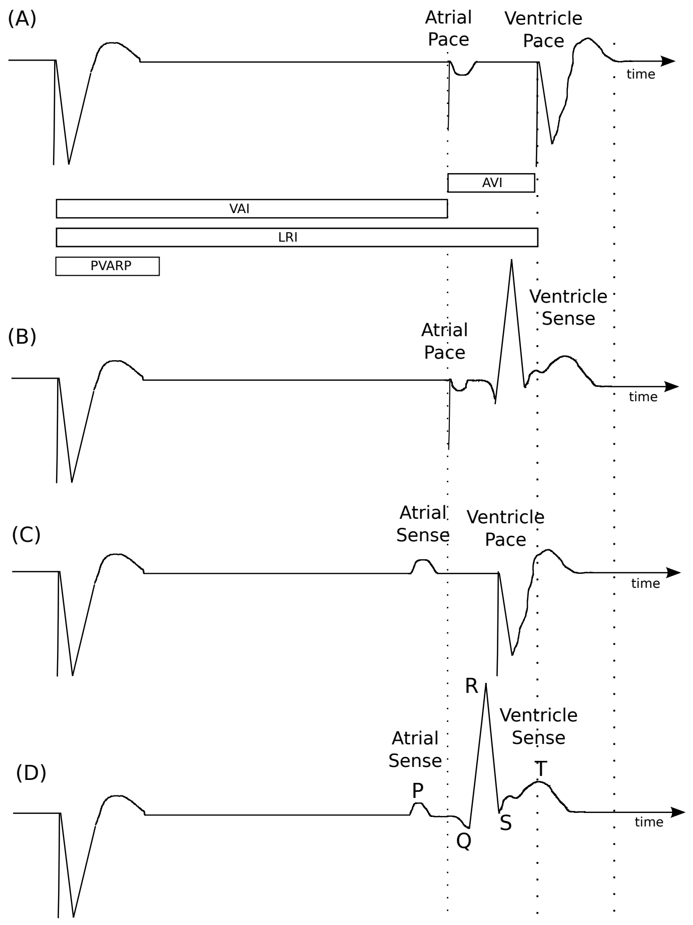


**Figure 3: The DDDR Pacing Scenarios**

Time goes left to right, and a flat line indicates no heart activity. A spike above the lines indicates intrinsic activity and a spike below the line indicates activity as a result of the action of the pacemaker. A rounded spike indicates activity in the atrial and a sharp spike indicates activity in the ventricle.

| Time Intervals | Purpose | Time in milliseconds |
|---|---|---|
| Length of a P wave ($T_P$) | Time during which intrinsic atrium activity must be sensed | 110 |
| Duration of pulse ($T_{pulse}$) | the time for which the pulse current must be maintained | 1 |
| Length of a QRS complex ($T_{QRS}$) | Time during which intrinsic ventricular activity must be sensed | 100 |
| Atrioventricular interval (AVI) | Provides time for ventricle to fill following an atrial contraction | 150 |
| Ventriculoatrial interval (VAI) | Ensures an atrial pulse following a ventricular pulse | 850 |
| Postventricular atrial refractory (PVARP) | Ensures atrium doesn't falsely sense ventricular activity | 350 |
| Mode Switching Interval (MSI) | Time between two atrial events used to change modes | 500 |

**Table 2: Example Timing Intervals in a Single Heart Beat**

VAI timeout. So the next AVI timeout started at that point and the ventricle sensor is allowed to sense ventricle activity only in the AVI period. In the DDIR mode, the next AVI timeout does not start until the full VAI period has expired, but the ventricle sensor is allowed to sense ventricle activity after intrinsic activity occurs in the atria chamber.

Fig. 4 gives a flow chart for the basic required operations of the DDDR mode. This gives an informal description of the requirements for the pacemaker operating in the DDDR mode. If a ventricle pulse has just been delivered, the pacemaker watches the atrial channel for a spontaneous P wave (intrinsic activity in the atrium). If the VAI times out, the pacer delivers a pacing pulse to the atrium; otherwise, the atrial output is inhibited. The pacemaker now watches the ventricle for a spontaneous QRS complex wave (indicating intrinsic activity in the ventricle). If it is detected, the ventricular pace is inhibited, otherwise the pacemaker delivers a pacing pulse. If the accelerometer detects a change in the patient's activity level, it changes the VAI timeout to compensate – thus speeding up or slowing down the heart beat.

The operating mode of the pacemaker changes under the following conditions [7, 14].

**From DDDR to DDIR** – if the time between two atrial events is less than some threshold, the Mode Switching Interval, on $X$ consecutive occasions.

**DDIR to DDDR** – if the time between two atrial events is greater than the same threshold on $Y$ consecutive occasions.

In this paper, we will use the times shown in Table 2 for the intervals of the pacemaker operating modes.

## 3. THE DESIGN OF THE SCJ SOFTWARE

Section 2.4 has illustrated the complexity of the required control requirements, and in particular the various changing time constraints (even without considering the impact of rate modulation). Consider the irregular heart beat illustrated in Figure 5. It shows that the required pacing activities are not regular enough to be controlled by a periodic activity. They are essentially aperiodic and time-triggered in the presence or absence of intrinsic heart activity.

Unfortunately, SCJ has limited support for a time-triggered programming model: only periodic activities, and no explicit support for a general timeout facility (that can be used to implement watchdog timers). SCJ is a subset of the Real-time Specification for Java (RTSJ), but does not support the RTSJ's one-shot timers, so these cannot be used to deal with the varying timeouts. One of the main recommendations from this study is that the SCJ should support a version of the RTSJ's one-shot timers – see section 5.

The intrinsic heart activities are also aperiodic in nature, and in SCJ must be polled for – as the pacemaker does not support report intrinsic activities via interrupts. Hence sensors will be monitored by periodic activities and pacers controlled by aperiodic activities.

The software architecture we have adopted is one where each SCJ operating mode is encapsulated in an SCJ mission which is comprised of several managed asynchronous event handlers. The periodic event handlers continually monitor the sensors and record the times at which significant heart activities occur. The handlers then detect the absence of intrinsic heart contractions and initiate the required artificial stimulus via SCJ's event-triggered aperiodic handlers. The program structure is illustrated in Fig. 6 – for each hardware sensor or actuator, there is an SCJ asynchronous event handler monitoring or controlling its activity.. The detailed flow charts for the functionality of the SCJ event handlers are shown in Figures 7 and 8. The code is available from `http://www.cs.york.ac.uk/circus/hijac/case.html`.

Table 2 summarizes the timing requirements for a pacemaker whose upper rate limit (the number of beats per minute of the heart) is required to be 120. Given that the time during which a P wave can be detected is 110 milliseconds, the period of the atrium sensor must be 55 milliseconds to ensure a P wave will not be missed. For simplicity, we use an implicit deadline model – that is, the deadline is equal to the period). Similarly, the period (and deadline) of the ventricular sensor must be 50 milliseconds. The aperiodic pacers are released by the sensors when intrinsic activity is not detected within the AVI and VAI durations. For safety, it is imperative to ensure that pulsing does not occur during a P wave.

In order to determine the appropriate times to initiate the pacing activities, the algorithm maintains the last time that activity occurred in both the atrium and the ventricle chambers. This activity is either intrinsic heart activity or induced activity as a result of pacing. Figure 7 shows the atrium sensor and pacer algorithms in detail. The atrium sensor does not read the physical sensor during the PVARP interval following ventricular activity or once the atrium activity has occurred. If intrinsic atrium activity is not detected during the VAI following the last ventricular activity then the pacer handler is released. Once released, the pacer handler ensures no recent atrium activity before initiating the pacing stimulus. Note that the time saved is for the initiation of the pacing current.

The structure of the ventricular sensor and pacer handlers is similar and given in detail in Figure 8. There are two additional points to make. Firstly, the test in the ventricle pacer ensures that no stimulus is initiation when a P wave might be occurring. Second, the algorithm checks to see whether a mode change should occur. If it finds a mode change is necessary, then it requests mission termination. Note that as the pacer will have already been released, the pacing will occur before the mission is terminated.

In both pacing algorithms, a "sleep" has been introduced in order to control the duration of the pacing current. The duration of the current is patient-dependent and might be significantly less than 2 milliseconds. In this case, a spinning delay can be used (the SCJ `nanoSpin` method) instead.

### 3.1 Handling Multiple Operating Modes

The software for many pacemakers is designed to operate in a single mode only. However, there are situations where there are requirements to switch between compatible modes. The example given in this paper is for switching between the DDDR and the DDIR modes. The heart is a robust systems and hence most of

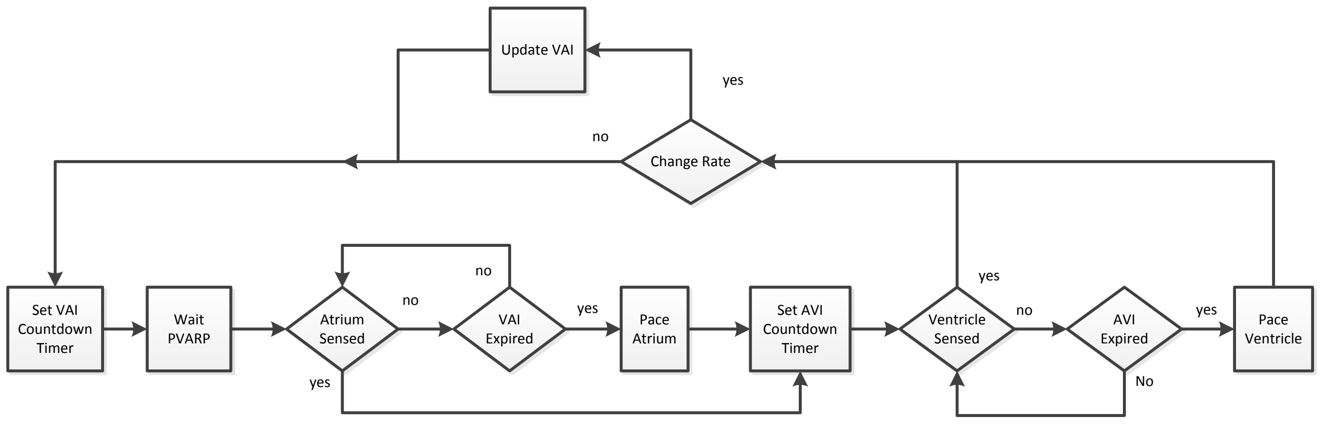


**Figure 4: The Required DDDR Pacing Cycle**

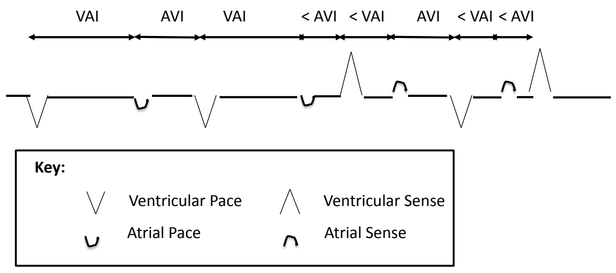


**Figure 5: An Irregular Heart Beat**

the deadlines given in the paper are soft rather than hard deadlines. Nevertheless, significant jitter of pulse activities or spurious pulses will inevitably lead to discomfort for the patient. Switching between the pacemaker's operating modes is one situation where jitter may be introduced.

There are essentially two ways in which mode changes can be handled in the SCJ implementation depending on the how the timing requirements are to be met during the mode switch.

1. Treat each mode as a separate SCJ mission and have the mission sequencer alternate between modes. The requirement for mode switch can be detected by the atrium sensor which can then request the termination of the current mission (as suggested in the previous subsection). The time of the last significant heart events can be maintained in immortal memory and these can be used to ensure the appropriate continuity of pacing and sensing activities. However, if the time taken for an SCJ implementation to terminate one mission and start the next is too long this may lead to patient discomfort. This might be increasing disconcerting if mode switching is very frequent. The main advantage of keeping the modes separate is that it maintains a clear distinction between the functionality of each mode.

2. Combine all the modes functionality into a single mission. This would reduce the potential lack of control during a mission change but may increase the complexity of the code – particularly if the functionality requirement is significantly different between modes.

For the two modes, given in this paper, the difference is minor and only involves when a timeout should be reset. In this situation, maintaining a clear separation may not be worth the overheads of mission termination and restart. So, we adopt solution 1 above in the code present in `http://www.cs.york.ac.uk/circus/hijac/case.html`

## Scheduling Model

The architecture depicted in Figures 6 to 8 has a simple scheduling model. The sensors are periodic and have a worst-case execution time equal to the longest path through the code. Their deadlines are equal to their period. The pacers are sporadic and have a minimum inter-arrival time determined by the AVI and VAI times. Again their worst-case execution times are simple to determine. However, the "sleeps" introduced to control the pacing currents must be accounted for when determining their response times, as should any local drift introduced by using a relative rather than an absolute delay.

The priority ordering of the sensor handlers can be based on rate monotonic priority ordering. The aperiodic (sporadic) handlers are given high priorities to reduce the jitter on pulse activities.

## 4. RAVENSCAR ADA

The design of the SCJ software architecture for the pacemaker is driven by the lack of one-shot timers in the SCJ specification. Ravenscar Ada, in contrast to SCJ, does support the full time-triggered programming model by providing a primitive timing-events

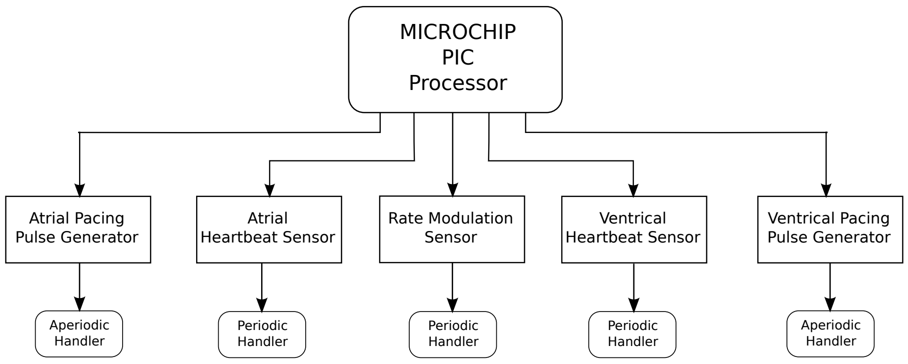
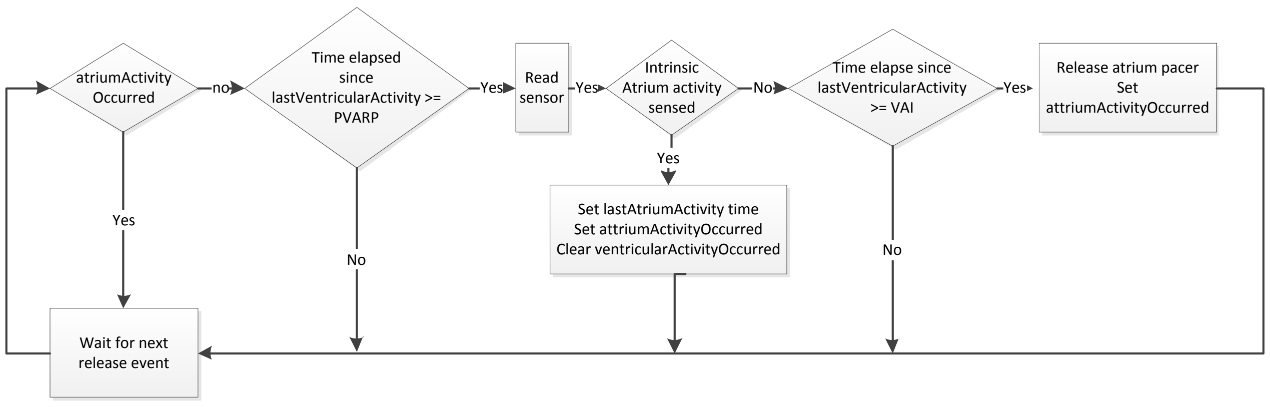
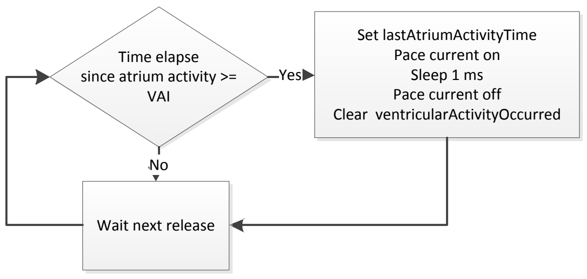


**Figure 6: Cardiac Pacemaker Architecture in SCJ**


atriumActivity Occurred
no
Time elapsed since lastVentricularActivity >= PVARP
Yes
Read sensor
Yes
Intrinsic Atrium activity sensed
No
Time elapse since lastVentricularActivity >= VAI
Yes
Release atrium pacer Set attriumActivityOccurred
Yes
Yes
Set lastAtriumActivity time Set attriumActivityOccurred Clear ventricularActivityOccurred
No
No
Wait for next release event

(a) Algorithm for the Atrium Sensor Periodic Handler


Time elapse since atrium activity >= VAI
Yes
Set lastAtriumActivityTime Pace current on Sleep 1 ms Pace current off Clear ventricularActivityOccurred
No
Wait next release

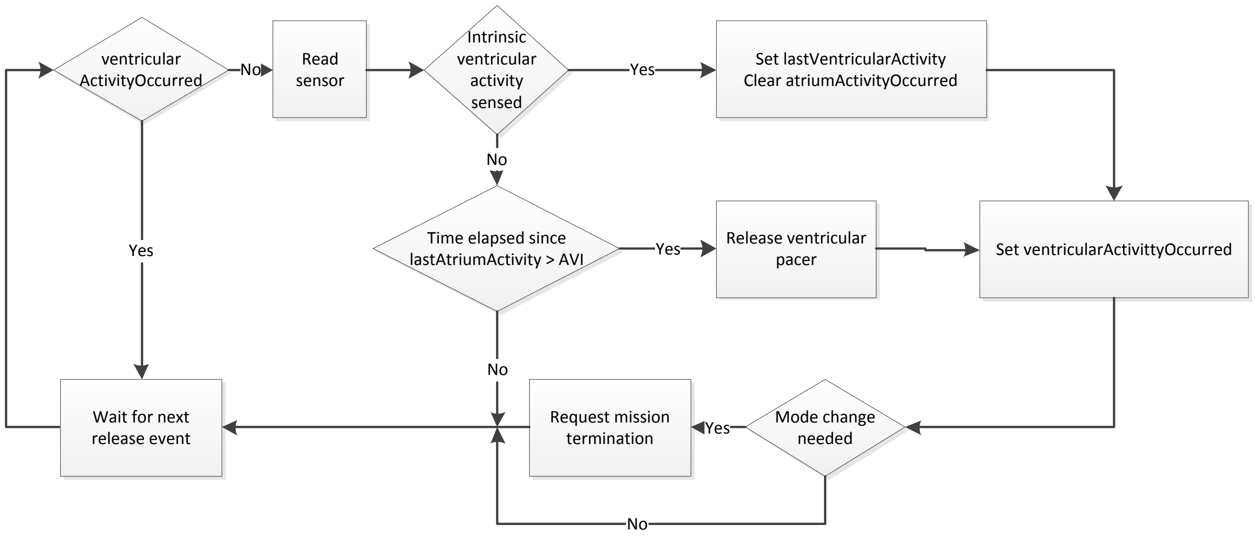
(b) Algorithm for the Atrium Pacer Aperiodic Handler

**Figure 7: The Atrium Sensor and Pacer Event Handling Code**

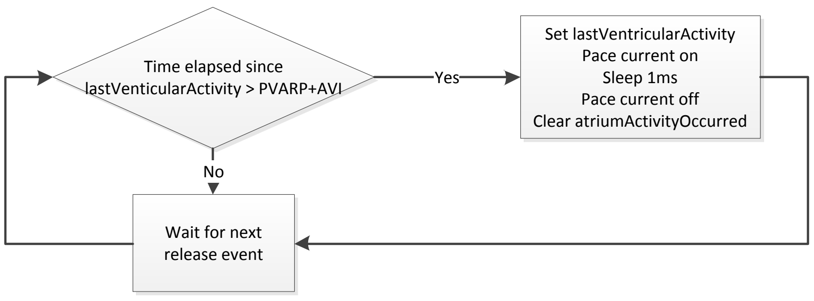
mechanism, which can be used to construct both periodic handlers and watch-dog timers. It is possible to adopt the same approach for a Ravenscar Ada solution as that adopted for SCJ: the sensors would be periodic Ada tasks, and the pacers would also be tasks waiting on an entry of a protected object for its release event. However, here we illustrate that with the use of a watchdog timer, a more optimized solution can be found, and one that requires very little run-time support. The focus is just on a simple DDD pacemaker with no rate modulation, although rate modulation is easy to implement.

The Ada solution (also found at `http://www.cs.york.ac.uk/circus/hijac/case.html`) more closely follows the approach shown in Figure 4 and uses Ada's timing events to control both the sensing and pacing activities. There are two main timing events: the first (`Watchdog`) is used to detect the absence of intrinsics activities and to control the pacing current, and the second (`Sensor_Readings`) is used to initiate the reading of sensor. A single protected object (`Timer`) is used for encapsulating the handlers for these timing events. This use of a single protected object removes the potential for race conditions to occur in the software.

The resulting program design is more efficient than its SCJ counterpart, with handlers only executing when control is needed. However, the structure of the solution is a little more complex. Furthermore, Ada requires that at least one task be present in the system to


ventricular ActivityOccurred
No
Read sensor
Intrinsic ventricular activity sensed
Yes
Set lastVentricularActivity Clear atriumActivityOccurred
No
Yes
Time elapsed since lastAtriumActivity > AVI
Yes
Release ventricular pacer
Set ventricularActivittyOccurred
No
Wait for next release event
Request mission termination
Yes
Mode change needed
No

(a) Algorithm for the Ventricle Sensor Periodic handler


Time elapsed since lastVenticularActivity > PVARP+AVI
Yes
Set lastVentricularActivity
Pace current on
Sleep 1ms
Pace current off
Clear atriumActivityOccurred
No
Wait for next release event

(b) Algorithm for the Ventricle Pacer Aperiodic Handler

**Figure 8: The Ventricular Sensor and Pacer Event Handling Code**

prevent program termination.

The details of the Ada solution are shown in Figure 9.

## Scheduling Model

The handlers for timing events in Ada are called from the Ada run-time clock interrupt handling code. Hence, no Ada tasks are actually required. However, the Ada language designers did not envisage pure time-triggered systems being implemented in Ada using just timing events. Hence the program termination rules are defined in terms of tasks. Consequently, it is necessary to introduce a dummy task in order to keep the program alive.

The pacemaker is a reactive system, and the solution adopted is similar to what might be expected from a synchronous programming language such as Esterel. As all code is run at interrupt-level, it is imperative to keep the handling code as simple and as short as possible so that the computation can be completed before another timing event needs to be set. The `Atrium_Pace_On`, `Atrium_Pace_Off`, `Ventricle_Pace_Off` and `PVARP_-Timeout` handlers are mutually exclusive events. Hence the requirements are for:

- `Atrium_Pace_On` to be completed before the pulse duration (Tpulse) has expired (1 millisecond – see Table 2) when `Atrium_Pace_Off` needs to be called. The code in the handler is simply one actuator operation and the setting of one timing event.

- `Atrium_Pace_Off` to be completed within safety margins for the the pulse duration (and before the AVI (150 milliseconds)). The code in the handler is simply one actuator operation and the setting of one timing event.

- `Ventricle_Pace_On` to be completed before the pulse duration (Tpulse) has expired (1 millisecond – see Table 2) when `Ventricle_Pace_Off` needs to be called. The code in the handler is simply one actuator operation and the setting of one timing event and the canceling of another.

- `Ventricle_Pace_Off` to be completed within safety margins for the the pulse duration (and before the PVARP duration (250 milliseconds)). The code in the handler is simply one actuator operation and the setting of one timing event.

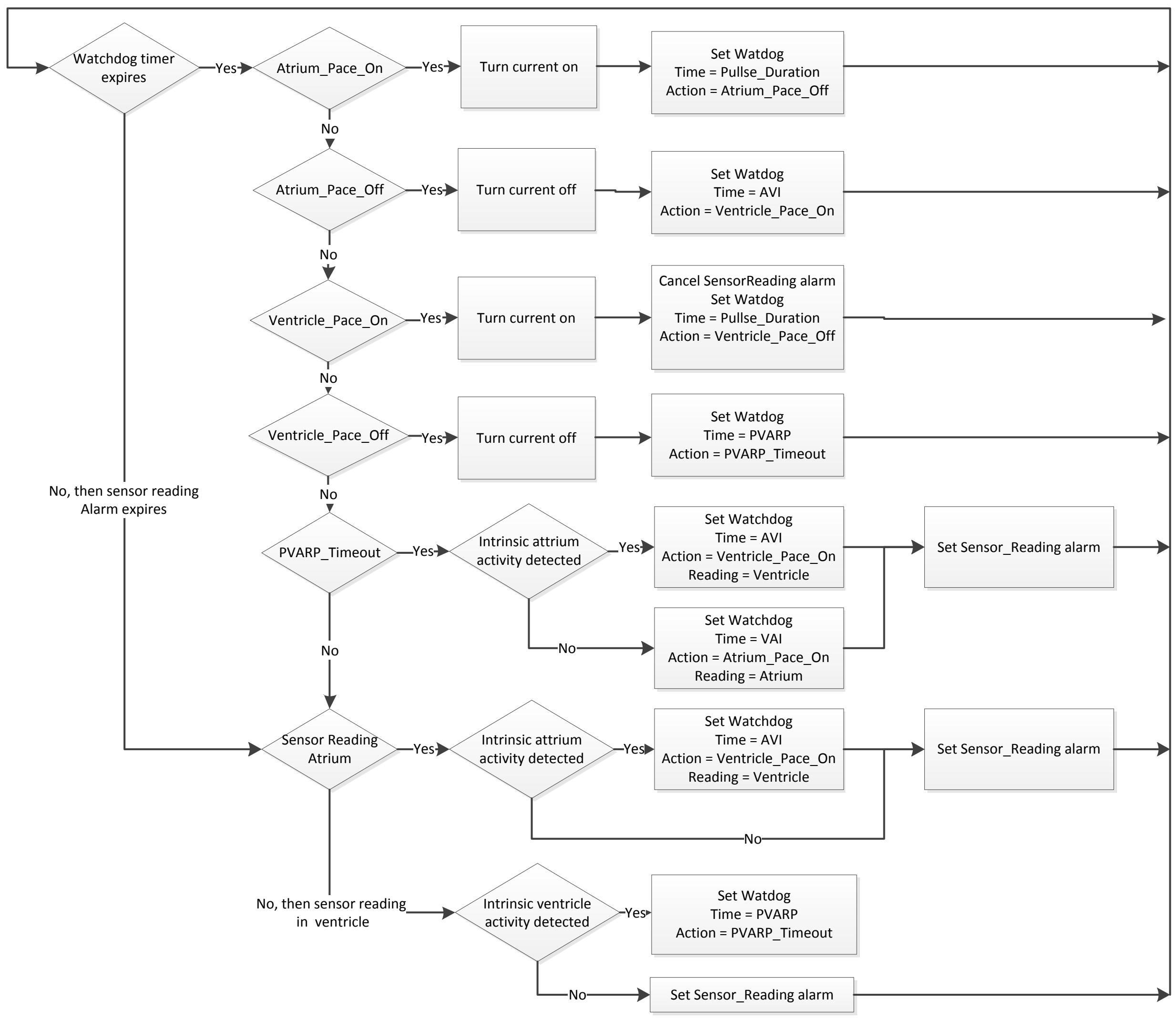


**Figure 9: Cardiac Pacemaker Architecture in Ada**

- `PVARP_Timeout`to be completed before the VAI duration (850 milliseconds). The maximum code in the handler is simply one sensor operation and the setting of two timing event.

The sensor reading handlers are similarly simple, mainly consisting of one sensor reading operation and at most two settings of timing events. The asynchronous relationship between the watchdog and sensor-reading timing events means that it is possible for one event to want to fire whilst the other event is being handled. Hence, the response time of each handler must include a blocking time equal to the maximum handling timing of the other event.

## 5. CONCLUSION AND FUTURE WORK

In this paper we have considered the design and implementation of a cardiac pacemaker in both Safety-Critical Java and the Ravenscar Profile for Ada. Our contributions are as follows:

1. The identification of a realistic case study whose control requirements are complex and do not conveniently fit the essentially periodic programming paradigms that are encapsulated by SCJ's and Ravenscar Ada's task model. Since SCJ is an emergent technology, there are currently very few studies available in the public domain. We are aware only of those in [15, 9, 1].
2. A discussion of the repercussions of the current draft release of the SCJ specification ([17]) not providing explicit support for watchdog timers.

   Their absence constrains the possible software architectures that can be produced. The approach adopted here is to record the time by which significant events must occur, and then poll periodically to detect their absence. As a result of this work, the final release of the SCJ will contain support for one-shot timers. This will allow a wider range of solutions to be considered, including one similar to the Ada solution presented in this paper.
3. The identification of an "ease of use" issue with using Ravenscar Ada to implement purely reactive systems. We use two

timers: the first is essentially a watchdog timer and the second is used to constrain the periods during which the sensors should be read.

The Ada designers clearly have not considered the full implications of using only timing events to implement reactive systems, as the program termination rules are specified in terms of tasks. As a consequence, it was necessary to include a dummy task into our system to avoid premature termination. Whilst this is an easy work-around, it is rather inelegant. We intend to raise this problem at the next International Workshop on Real-Time Ada Issues, where problems with the language are often discussed before solutions are proposed.

The algorithms in this paper have been implemented and tested with a simulator along with an RTSJ version that make use of one-shot timers. The basic algorithms that available at `http://www.cs.york.ac.uk/circus/hijac/case.html`. In future, we plan to perform formal verification of the cardiac pacemaker models using *Circus* [26, 27], and investigate the refinement of these *Circus* models into SCJ.

## 6. REFERENCES

[1] Austin Armbruster, Jason Baker, Antonio Cunei, Chapman Flack, David Holmes, Filip Pizlo, Edward Pla, Marek Prochazka, and Jan Vitek. A real-time java virtual machine with applications in avionics. *ACM Trans. Embed. Comput. Syst.*, 7(1):5:1–5:49, December 2007.

[2] S. Serge Barold, Roland X. Stroobandt, and Alfons F. Sinnaeve. *Cardiac Pacemakers Step by Step*. Futura Publishing, 2004. ISBN 1-4051-1647-1.

[3] A Burns and A. J. Wellings. *Concurrent and Real-Time Programming in Ada*. Cambridge University Press, 2007.

[4] Alan Burns. The Ravenscar profile. *ACM Ada Letters*, XIX(4):49–52, Dec 1999.

[5] P. Dibble, R. Belliardi, B. Brosgol, D. Holmes, and A.J. Wellings. The real-time specification for Java: Version 1.02. Available at `http://www.rtsj.org/specjavadoc/book\_index.html`, last accessed August 2012.

[6] Kenneth A. Ellenbogen and Mark A. Wood. *Cardiac Pacing and ICDs*. 4th Edition, Blackwell, 2005. ISBN-10 1-4051-0447-3.

[7] Stabile Giuseppe, Simone Antonio De, and Romano Enrico. Automatic mode switching in atrial fibrillation, July 2005.

[8] Artur Gomes and Marcel Oliveira. Formal specification of a cardiac pacing system. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 692–707. Springer Berlin / Heidelberg, 2009.

[9] H. Hamza and S. Counsell. Simulation of safety-critical, real-time Java: A case study of dynamic analysis of scoped memory consumption. *Simulation Modelling Practice and Theory*, vol 25, doi:10.1016/j.simpat.2012.02.011:172–189, 2012.

[10] Aaron Hesselson. *Simplified Interpretations of Pacemaker ECGs*. Blackwell Publishers, 2003. ISBN 978-1-4051-0372-5.

[11] C.A.R. Hoare, Jayadev Misra, Gary T. Leavens, and Natarajan Shankar. The verified software initiative: A manifesto. *ACM Comput. Surv.*, 41(4):1–8, 2009.

[12] Tony Hoare. The verifying compiler: A grand challenge for computing research. *J. ACM*, 50(1):63–69, 2003.

[13] James Hunt and Kelvin Nilsen. Safety-Critical Java: The mission approach. In M. Teresa Higuera-Toledano and Andy Wellings, editors, *Distributed, Embedded and Real-time Java Systems*, incollection 9, pages 199–233. Springer US, 2012.

[14] Carsten W. Israel. Automatic mode switching in atrial fibrillation. *Pacing and clinical electrophysiology : PACE*, 25(3):380–393, March 2002.

[15] Tomas Kalibera, Jeff Hagelberg, Filip Pizlo, Ales Plsek, Ben Titzer, and Jan Vitek. Cdx: a family of real-time java benchmarks. In *Proceedings of the 7th International Workshop on Java Technologies for Real-Time and Embedded Systems*, JTRES '09, pages 41–50, New York, NY, USA, 2009. ACM.

[16] Bharat K. Kantharia and Steven P. Kutalek. Optimal programming of rate modulation functions. *Cardiac Electrophysiology Review*, 3:53–55, 1999. 10.1023/A:1009935600754.

[17] D. Locke, B.S. Andersen, B. Brosgol, M. Fulton, T. Henties, J.J. Hunt, J.O. Nielsen, K. Nilsen, M. Schoeberl, J. Tokar, J. Vitek, and A.J. Wellings. Safety-critical Java technology specification, public draft. Available at `http://www.jcp.org/en/jsr/detail?id=302`, 2011.

[18] Hugo Daniel Macedo, Peter Gorm Larsen, and John Fitzgerald. Incremental development of a distributed real-time model of a cardiac pacing system using vdm. In *Proceedings of the 15th international symposium on Formal Methods (FM'08)*, Lecture Notes in Computer Science, pages 181–197, Berlin, Heidelberg, 2008. Springer-Verlag.

[19] Valerio Panzica La Manna, Andrea Tommaso Bonanno, and Alfredo Motta. Poster on a simple pacemaker implementation. ACM, May 2009.

[20] Dominique Méry and Neeraj Kumar Singh. Automatic Code Generation from Event-B Models. In *Proceedings of the 2011 Symposium on Information and Communication Technology (SoICT)*, Hanoi, Vietnam, 2011. ACM, ACM International Conference Proceeding Series.

[21] Dominique Méry and Neeraj Kumar Singh. Functional Behavior of a Cardiac Pacing System. *International Journal of Discrete Event Control Systems*, 1(2):129–149, 2011.

[22] Boston Scientific. Pacemaker system specification, technical report. http://www.cas.mcmaster.ca/sqrl/SQRLDocuments/PACEMAKER.pdf, 2007.

[23] Luu Anh Tuan, Man Chun Zheng, and Quan Thanh Tho. Modeling and verification of safety critical systems: A case study on pacemaker. *Secure System Integration and Reliability Improvement*, pages 23–32, june 2010.

[24] Andy Wellings and Minseong Kim. Asynchronous event handling and Safety-Critical Java. *Concurrency and Computation: Practice and Experience*, vol 23, doi: 10.1002/cpe.1756, 2011.

[25] Jim Woodcock. First steps in the verified software grand challenge. *IEEE Computer*, 39(10):57–64, 2006.

[26] Jim Woodcock and Ana Cavalcanti. A concurrent language for refinement. In Andrew Butterfield, Glenn Strong, and Claus Pahl, editors, *IWFM*, Workshops in Computing. BCS, 2001.

[27] Jim Woodcock and Ana Cavalcanti. The semantics of circus. In Didier Bert, Jonathan Bowen, Martin Henson, and Ken Robinson, editors, *ZB 2002:Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science*, pages 184–203. Springer Berlin /

Heidelberg, 2002. 10.1007/3-540-45648-1_10.

[28] Members Writing Committee, Andrew E. Epstein, John P. DiMarco, Kenneth A. Ellenbogen, III Estes, N.A. Mark, Roger A. Freedman, Leonard S. Gettes, A. Marc Gillinov, Gabriel Gregoratos, Stephen C. Hammill, David L. Hayes, Mark A. Hlatky, L. Kristin Newby, Richard L. Page, Mark H. Schoenfeld, Michael J. Silka, Lynne Warner Stevenson, and Michael O. Sweeney. ACC/AHA/HRS 2008 Guidelines for Device-Based Therapy of Cardiac Rhythm Abnormalities. *Circulation*, 117(21):2820–2840, 2008.