Modelling the Pacemaker in Event-B: Towards Methodology for Reuse

Michael Poppleton, Abdolbaghi Rezazadeh

School of Electronics and Computer Science,
University of Southampton, Highfield,
Southampton SO17 1BJ, UK,
mrp@ecs.soton.ac.uk, ra3@ecs.soton.ac.uk

Abstract. The cardiac pacemaker is one of the system modelling problems posed to the Formal Methods community by the *Grand Challenge for Dependable Systems Evolution* [11]. The pacemaker is an intricate safety-critical system that supports and moderates the dysfunctional heart's intrinsic electrical control system. This paper focusses on (i) the problem (requirements) domain specification and its mapping to solution (implementation) domain models, (ii) the significant commonality of behaviour between its many operating modes, emphasising the potential for reuse, and (iii) development and verification of models.
We introduce the problem and model three of the operating modes in the problem domain using a state machine notation. We then map each of these models into a solution domain state machine notation, designed as shorthand for a refinement-based solution domain development in the Event-B formal language and its RODIN toolkit.

1 Introduction

In 2007, the SQRL lab at McMaster University hosted the Pacemaker case study[1] of the international *Grand Challenge for Dependable Systems Evolution* [11]. This challenge to the software development and verification community was proposed by Tony Hoare in 2005. The pacemaker is a particularly demanding device in engineering terms: although the demands of control systems development are well known, the task of the pacemaker is not to control the dysfunctional heart, rather to support and correct its internal electrical control system.

A number of researchers have investigated the pacemaker as specified by Boston Scientific [18]. The Z specification of [8] is concrete, defining pulse generator, sensor, timing and battery power and associated behaviours, at one level of abstraction. Verification comprises proving initialisation consistency and operation totality with the ProofPower-Z tool. [14] use VDM++, a VDM variant with object-oriented and concurrent modelling, in an incremental process of producing abstract, sequential, concurrent, and finally distributed real-time models. These are not formal refinements, and validation is not formal, but performed by by scenario-based testing tools.

[16], based on previous pacemaker modelling proposes a scheme for certifiable development based on informal requirements structuring, formalization and proof, and

[1] http://sqrl.mcmaster.ca/index.html

then validation and animation. These authors' work [15] is closest to ours, using the refinement-based method of Event-B and its Rodin toolkit. Each mode undertaken is separately modelled and refined, again starting with the rather concrete concerns of timing, and sensing and pacing implementation. One group [10] at least have modelled the heart, to act as heart simulator and test-case generator for pacemaker models; a co-simulation approach is also planned for the VDM++ work [14].

The electrocardiology domain where the pacemaker requirements are articulated, is complex, e.g. [3, 7]. Initially, we chose to model required behaviour as state machines over the state and transition concepts arising in the problem domain; we call these *problem* models. Each problem model is "flat", i.e. at a single abstraction level. The modelling of various operating modes revealed considerable commonality of behaviour, and thus potential for model reuse, between modes.

In parallel, initial Event-B (solution, or implementation domain) modelling was undertaken, illustrated by a shorthand state machine representation - we call these *solution* models. In this activity, requirements are layered into the development through stepwise refinements. For dual-channel modes, the Event-B work suggested that a model design of two communicating state machines - one each for atrial and ventricular channels - was simple in structure and could capture much of the commonality of behaviour. Whereas the initial problem model was a single state machine, we then produced a version corresponding to the communicating two-machine solution domain state machines. The abstract solution-domain state machine is refined through more detailed state machines, implementing the problem domain requirements in layers.

We have modelled the five modes AAI, VVI, VDD, DVI and DDD but only discuss the first three here. These were selected for both learning and reuse reasons - AAI and VVI are single-channel models, the latter three more complex dual-channel models. DDD is effectively a composition of AAI, VDD and DVI. This work is distinct from other approaches in the following aspects: (i) the distinction between problem-space (electrocardiology) and solution-space (Event-B) state machine models, and the value of a "co-modelling" approach, (ii) the emphasis on reuse, (iii) the use of refinement to maintain abstraction, and incorporate timing some way down the chain, using the design pattern of [5].

In sections 2, 3 we provide background on Event-B and electrocardiology respectively. Section 4 introduces problem models for the simple, single-channel modes AAI and VVI. It then introduces the corresponding solution models in state machine and Event-B formats. Section 5 does the same for VDD mode. Section 6 concludes and also describes ongoing and future work.

2 Event-B Basics

This section is a précis of parts of [1, 17], which define the Event-B language.

Event-B is designed for long-running *reactive* hardware/software systems that respond to stimuli from user and/or environment. The set-theoretic language in first-order logic (FOL) takes as semantic model a transition system labelled with event names. The correctness of a model is defined by an invariant property, i.e. a predicate, or constraint, which every state in the system must satisfy. More practically, every event in the system

must be shown to preserve this invariant; this verification requirement is expressed in a number of *proof obligations* (POs). In practice this verification is performed either by model checking or theorem proving (or both).

For modelling in Event-B the two units of structuring are the *machine* of dynamic variables, events and their invariants, and the *context* of static data of sets, constants and their axioms. Every machine may *see* one or more contexts. The unit of behaviour is the *event*. An event e acting on (a list of) state variables v, subject to *guard* $G(x, v)$ over local variable x and *action* $E(x, v)$, has syntax

$$e \;\widehat{=}\; \textbf{any}\ x\ \textbf{where}\ G(x, v)\ \textbf{then}\ v := E(x, v)\ \textbf{end}$$

That is, when the state is such that the guard is true, this enables the action, or state transition defined by $E(x, v)$ for some value of x.

An event e works in a model (comprising a machine and at least one context) with constants c and sets s subject to *axioms* (properties) $P(s, c)$ and an *invariant* $I(s, c, v)$. Thus the event guard G and assignment with before-after predicate E take s, c as parameters. Two of the consistency proof obligations for event e are feasibility and invariant preservation.

The refinement of a context is simply its elaboration, by the addition of new sets, constants and axioms. The refinement of a machine includes both data and algorithm refinement: variables v are replaced or supplemented (or both) by new ones w. Existing events are transformed to work on the new variables, and new events can be defined; that is, the behaviour of an abstract event e can be refined by some sequence of new events. The new behaviour will usually reduce nondeterminism. When model $N(w)$ refines $M(v)$, it also has an invariant $J(s, c, v, w)$ which can include M's variables v. This "gluing invariant", or refinement relation, has the function of relating abstract variables v to concrete ones w mathematically. [1] defines further proof obligations for refinement.

The Rodin toolkit [2] for Event-B is a rich set of tools for model construction, analysis, verification and code generation; it comprises editors, syntax- and type-checking, proof obligation generator, proof manager, model-checker, animators, and provers. Also, composition, decomposition and other support plugins are available[2].

3 Electrocardiology and pacemaker basics - M

The essential function of the heart - its beating in a purposeful pattern - is determined by the pattern of electrical depolarization and associated contraction of specific heart tissues. Fig. 1 shows the heart's electrical system, comprising three elements [13]: (i) the sinoatrial (SA) node, in the right atrium, (ii) the atrioventricular (AV) node, on the interatrial septum close to the right ventricle, and (iii) the His-Purkinje fibres, in the ventricle walls. A standard tool for measuring this electrical activity is the electrocardiogram (ECG), a graphic trace over a period of time taken noninvasively on the chest. Fig. 2 shows a typical ECG, with a P wave (associated with atrial contraction), a QRS complex (associated with ventricular contraction) and a T wave (associated with ventricular relaxation).

[2] http://www.event-b.org/

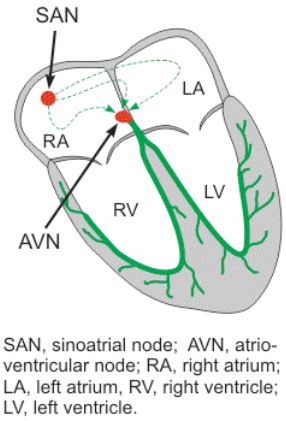

Fig. 1. Electrical structure of the heart [12]

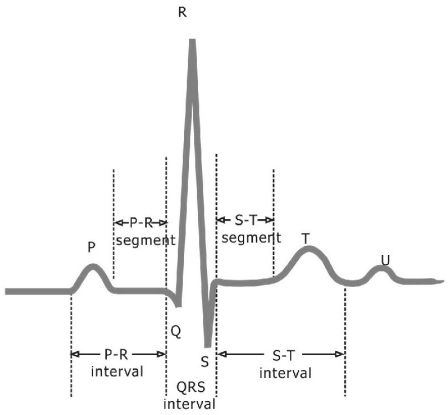

Fig. 2. ECG [6]

The control cycle of a normally functioning heart is as follows; we indicate the ECG wave corresponding to each stage: (1) A signal arrives at the SA node, or the heart's "natural pacemaker", causing contraction of the atria (P), (2) The ventricles fill, and then the signal arrives at the AV node (the line segment between P and Q), (3) The signal continues into the ventricular tissues through the His-Purkinje fibre system, causing ventricular contraction, where the right ventricle contracts a brief interval after the left (Q when signal is at His, R when signal is in the Purkinje system and left ventricle contracts, S when right ventricle contracts), (4) The ventricular walls relax and the ventricles expand (T).

To monitor and correct the dysfunctional heart, the pacemaker is a sealed battery-powered electronic device implanted in the chest wall [3]. Depending on the design (single or dual electrode, thus single- or dual-chamber operation) and operating mode, the pacemaker can sense depolarization of atria and ventricles, and can deliver pacing signals to atria and ventricles. It is a device that must support and moderate the intrinsic control system of the heart, given the diagnosed dysfunction in that control system. Modern pacemakers are reconfigurable (by resetting operating parameter values) and reprogrammable (by changing operating mode) in service, in the patient's body.

The pacemaker operating mode is described by a five-letter code - of which only the first three are relevant to this paper - called the NBG code for pacing nomenclature [7]. The first position refers to the chamber(s) in which stimulation, or pacing, occurs: A for atrium, V for ventricle, D for both chambers. The second refers to the chamber(s) in which sensing occurs: the code is as for the first position. The third position refers to the mode of sensing, i.e. the device's behaviour in response to a sensed event. I indicates that anticipated pacing is inhibited by a sensed event, T that pacing is triggered by a sensed event (e.g. V pacing a suitable time after a sensed A event). D indicates a dual mode (I for one chamber, T for the other) in a dual-chamber pacemaker.

Refractory periods are important in cardiac pacing: once depolarized, heart tissues are refractory (unresponsive) to electrical stimulation for a certain period, until they are

repolarized. Separately, a pacemaker may employ refractory periods during which it does not sense on A or V.

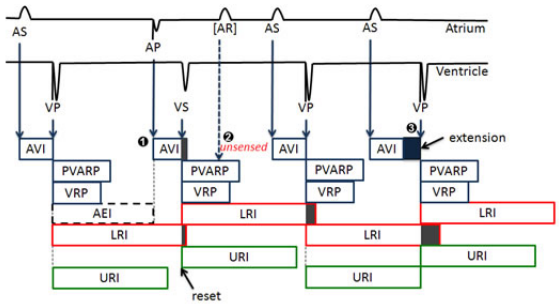

Fig. 3. Pacemaker timing intervals - DDD [10]

In this paper we consider only single-chamber modes AAI, VVI and the dual-chamber mode VDD. The most sophisticated mode DDD senses both A and V, and paces these chambers in inhibited manner i.e. only when no intrinsic pulse is sensed in a chamber. DDD uses six time intervals[3] - see Fig. 3:

1. LRI (lower rate interval): maximum interval between two ventricular beats, corresponding to the minimum heart rate
2. AVI (atrioventricular interval): interval from atrial pace or sensed event to ventricular pace
3. AEI (atrial escape interval): interval from from ventricular pace or sensed event to atrial pace
4. PVARP (post-ventricular refractory period): the period after a sensed or paced ventricular event, during which the atrial sensing circuit is refractory. This is to prevent *crosstalk*, where the atrial sensor detects and misinterprets a ventricular signal
5. VRP (ventricular refractory period): the period after a sensed or paced ventricular event, during which the ventricular sensing circuit is refractory
6. URI (upper rate interval): minimum interval between two ventricular beats, corresponding to the maximum heart rate

4 AAI to VVI

In this section we introduce the simplest single-chamber modes AAI and VVI. We describe the problem domain notation used and give models for these modes. Section 4.1 then shows the corresponding solution space state machine, as shorthand for an Event-B model, to be constructed manually or using a tool such as the RODIN UML-B plugin [19].

[3] All intervals apart from URI are used by modes discussed in this paper.

The electrocardiology sources [7, 3] for this work describe the observable behaviour of the pacemaker in terms of a single process of events of starting/stopping refractory and sensing states, detecting an intrinsic A or V signal, and pacing A or V. This process is naturally characterized in context of a timing diagram such as Fig. 3.

We produced single state machine models for AAI and VVI - see Figs. 4, 5. Each of these models is *flat*, containing all requirements information at one abstraction level, as opposed to the staged refinement process in Event-B. In parallel, abstract Event-B models were being written for these modes in a style that could be described in shorthand in state machine form.

In *atrial inhibited pacing* (AAI), only the atrial channel is paced and sensed - see Fig. 4. An atrial event, paced or sensed, initiates the atrial refractory period (ARP), during which time no pacing or sensing occurs. This event also initiates the LRI. After the ARP expires, sensing is again activated. If no atrial event is sensed before the LRI expires, the atria are paced. This cycle is insensitive to ventricular events as the pacemaker has no information about the ventricular channel.

For this model we consider only time interval events (start, elapse, stop) and pacemaker actions (start/ stop sensing, pace). We model the pacemaker as a long-running, cyclic process, or sequence of events, that occur in between quiescent states:

- Refractory Pacemaker is insensitive to any intrinsic heart signal
- Monitoring Pacemaker is sensing on this channel
- SensedIntSignal Pacemaker has just sensed an intrinsic signal on this channel; state represents the time elapsed during the sensing of an intrinsic signal
- Pacing Pacemaker is pacing on this channel; state represents the time taken to action the pace

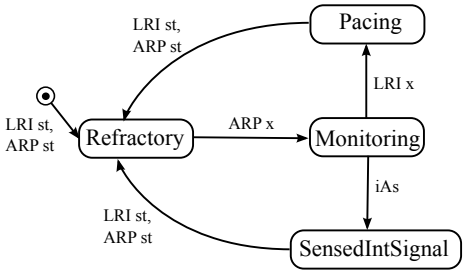

Fig. 4. Atrial inhibited pacing - AAI

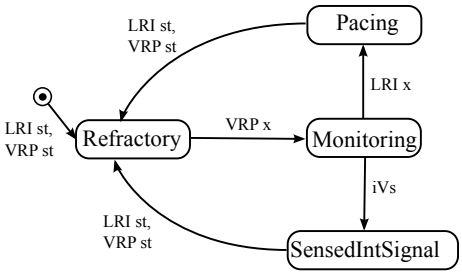

Fig. 5. Ventricular inhibited pacing - VVI

We annotate each arrow with the relevant problem domain event(s): this should be understood as a single state transition that models these events happening together atomically and instantaneously. If an arrow has more than one event, these are denoted comma-separated. We abstract away the "sensing on/off" events, regarding them as internal to state Monitoring. Similarly we abstract away/hide the "pacing" event in state Pacing.

We describe AAI through Fig. 4. The start event results in state Refractory as a convenient state, where an atrial event has just occurred. The LRI and ARP intervals are started on entry to this state. After ARP expires ARP x, sensing (Monitoring in this model) commences. The next event is either an intrinsic atrial signal iAs, or the expiry of the LRI time interval LRI x, the latter resulting in a atrial pace. Both the corresponding states Sensed_IntSignal and Pacing are followed by events restarting the LRI and ARP intervals LRI st, ARP st, and then once more entering Refractory.

Our first observation of the opportunity for model reuse is that the problem model for VVI corresponding to Fig. 4 is produced by trivial refactoring into Fig. 5, replacing the ARP with the ventricular refractory period VRP, and iAs with intrinsic ventricular signal iVs. VVI behaviour is identical to AAI, on the complementary chambers to the atria, subject to complementary time intervals. A similar trivial refactoring is applied to the corresponding solution model Fig. 8.

4.1 AAI - solution domain Event-B development

Refinement-based Event-B modelling is a complementary approach to the flat process modelling of the previous section. With reuse across modes in mind, we start from a more abstract level, and layer features into our Event-B model in a stepwise manner with refinement. We will thus construct an Event-B development including all requirements of the problem model of the previous section.

An abstract view of the pacemaker is that it monitors the heart condition, and at certain times it may intervene by issuing a pacing pulse in the atrium or ventricle or both. The precise details of how the pacemaker reacts to the heart condition depends on the operation mode and the actual condition of the heart. For a basic mode such as AAI, the three distinguishable states are *Monitoring*, *Pacing* and *Refractory*, the latter being the quiescent state described in the previous section. Using these three basic states and relevant transitions, an abstract model of a pacemaker is presented in Fig. 6.

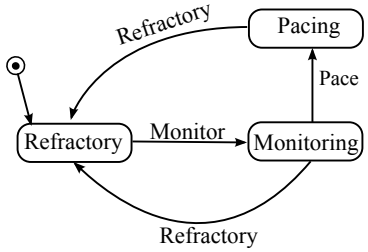

Fig. 6. Solution space state machine: AAI

This state machine is directly translated to the Event-B model of Fig. 7 where the state is represented by a state variable. The main value of this abstract model is that it provides a basis for an understanding and verification of the pacemaker functionality without the need to bring in complex timing aspects in the very early stage of modelling.

This abstract model will later be elaborated through stepwise refinement in Event-B to ultimately include the detailed requirements of Fig. 4.

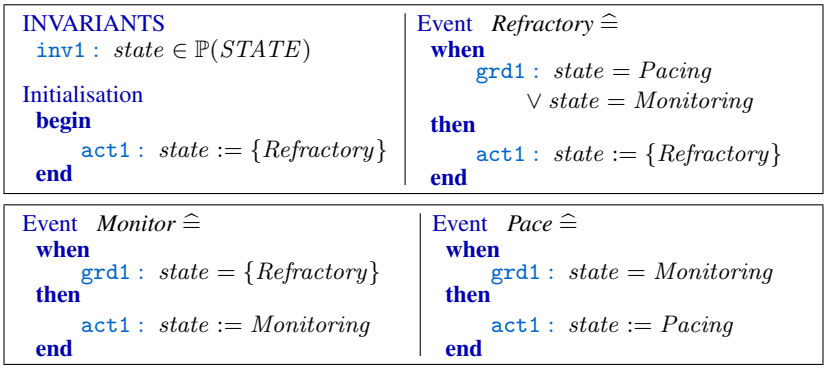

Fig. 7. First Level Specification

As indicated in Figs. 6 and 7, the initial state is set to *Refractory*. From this initial state the next state is *Monitoring*, representing the interval that the pacemaker monitors the heart condition. Note that at this stage we have not specified how the monitoring process is performed. This deliberately has been left out to allow us to have a very abstract view of the pacemaker functionality. If the system is in the *Monitoring* state the guards of both *Refractory* and *Pace* events are valid, indicating that a nondeterministic choice between these two events is enabled. This means that at this stage we do not specify under exactly what conditions the pacing should happen. If the *Pace* event happens then the following enabled event is the *Refractory* event, demonstrating the cyclic behaviour of the system.

The next step is to address the two issues that we left out in the abstract model: specifying how the pacemaker performs the monitoring task, and secondly specifying under what condition pacing can take place. A state machine representation of this refinement is presented in Fig. 8 and the textual Event-B refinement in Fig. 9. This state machine corresponds to the problem model in Fig. 4; here the state is represented by a state variable, and the the problem domain conceptual events become Event-B state transition events, themselves placeholders for the time interval management at lower refinement levels. The only structural changes to the graph are two modelling details driven by the Event-B coding: elaboration of the Pacing state into sub-states, and introducing Sensing as an explicit event.

In order to monitor the heart condition the pacemaker needs to sense intrinsic heart signals. To model this, the *Monitoring* state is refined to two sub-states, namely *StartMonitoring* and *SensedIntSignal* - see Fig. 8. A *Sensing* event is introduced that can repeatedly happen during the *StartMonitoring* sub-state. If any intrinsic signal is detected, the *intrinsic_signal* flag is set to true in the *Sensing* event and this in turn enables the *SensedIntSignal* event, see Fig. 9. This event changes the state from *StartMonitoring*

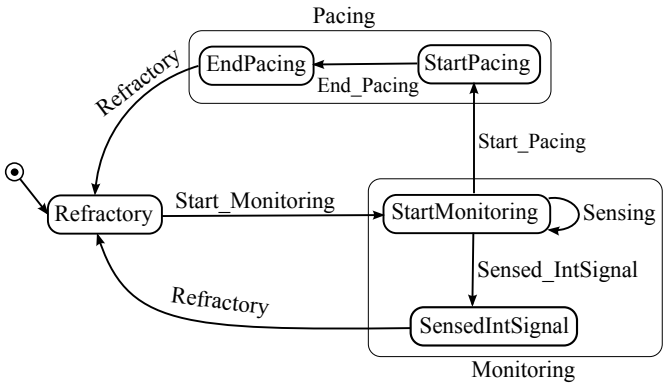

Fig. 8. Solution space state machine: AAI

to *SensedIntSignal*; a transitional state that can act as a placeholder for any extra evaluation of the intrinsic signal before moving to the *Refractory* state. On the other hand, if during *StartMonitoring* state, no intrinsic signal is sensed, then the pacemaker should eventually issue a pace signal. This situation is captured in the guards of *Start_Pacing* event, which makes the transition from *StartMonitoring* state to the *Pacing* state more deterministic than it was in the abstract level. Although the duration of the pace signal is very short, it nevertheless takes some time. To reflect this we have refined the *Pace* event of the abstract model into two events, *Start_Pacing* and *End_Pacing*, presented in Fig. 9. The *Start_Pacing* event is a direct refinement of the *Pace* event and the second event is new. The *pace_signa*l flag is set to *TRUE* in the *Start_Pacing* event to indicate that the output signal is active and set back to *FALSE* in the *End_Pacing* event to indicate the end of pacing period.

The first two levels of Event-B modelling used set-based representation of the state to enforce ordering between events without explicitly specifying any timing properties. The second refinement, partially presented in Fig. 10, introduces the exact timing intervals. Note that in this refinement we do not introduce any new events, therefore the state machine representation of this refinement is identical to that of Fig. 8. We have adapted the approach of [5] to model timing. This is a discrete time approach using a set of naturals to model future event times, that allows the clock to advance nondeterministically up to the next event time. Events in Event-B are atomic and their occurrences conceptually take no time; thus time elapses in each state. Variable *interval* is a singleton function mapping the current state to a time interval. In each event, the new state and interval are set. Here to maintain reusability we have used generic values. For example, the reader can see that in Fig 10 we have used $Refractory\_Period$ and $Alert\_Period$. These can be instantiated to ARP and LRI for AAI mode and to VRP and LRI for VVI mode respectively. Events include guards over these time intervals to ensure time elapses correctly before the system enters a new state and interval.

The guard of *Clock* event allows the clock to proceed up to the end of the current interval for the current state, after which a correct state transition is necessary for time

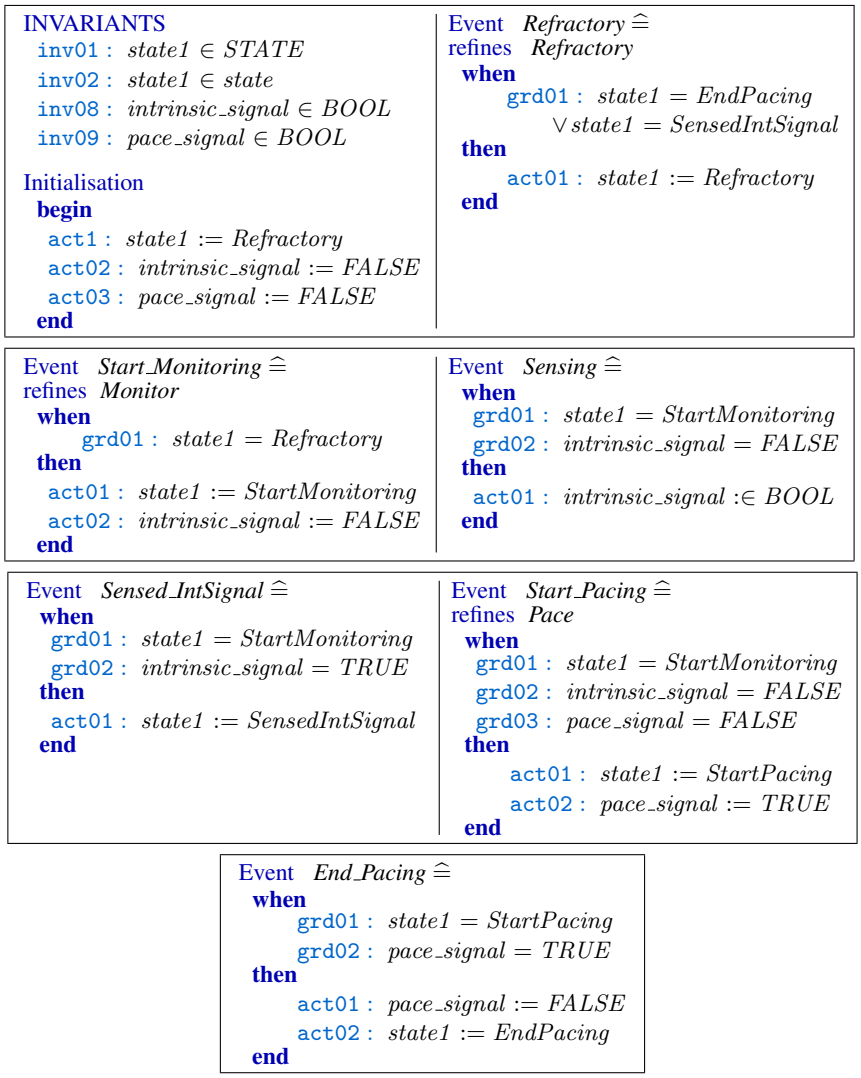

Fig. 9. First Refinement: Introducing Sensing and Pacing

to progress. The main advantage of this approach is that unlike other approaches such as [4], adding new events does not adversely enlarge the guard of *Clock*. Another advantage is that we can smoothly move from set-based sequencing of events to a more precise model based on discrete time intervals.

By this stage all timing requirements presented in Fig. 4 are encoded into the Event-B model. As indicated earlier, the Event-B models of this section can be used for either of AAI or VVI modes just by renaming events and intervals. This is another advan-

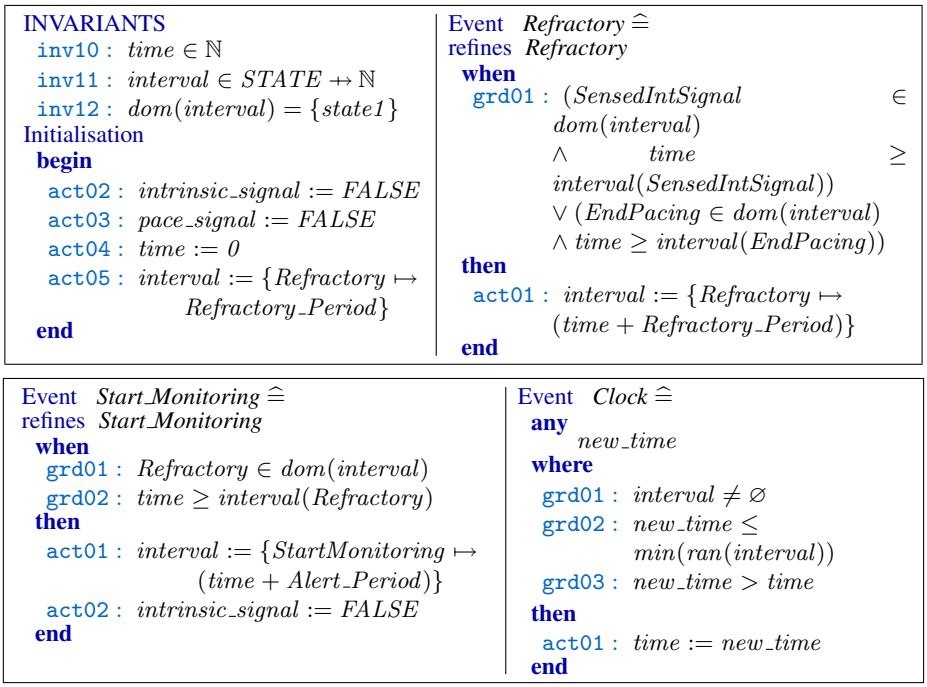

Fig. 10. Second Refinement: Introducing Timing Properties

tage of our approach. In the next section we demonstrate how by combining a subset of events and states from these two modes we can construct more complex models representing other modes of operation in pacemakers.

5 VDD

As in section 4 we give a problem model for VDD; section 5.1 gives in part the corresponding solution space state machine and Event-B models, to be constructed manually.

In VDD mode the ventricles are paced (V), both atrial and ventricular channels are sensed (D), and both inhibited and triggered behaviours (D) are present. VDD mode is known as *P-synchronous pacing*: V-pacing tracks, or follows, the P-wave (intrinsic atrial signal), and V-pacing is inhibited by an R-wave (intrinsic ventricular signal). We responded to the initial Event-B modelling by producing a problem model comprising two cyclic synchronised state machines for the atrial and the ventricular channels - we will call these A and V respectively, see Fig. 11.

Here we see more reuse opportunites. Firstly, each atrial model and each ventricular model comprises an extended subgraph over the four states of AAI (Fig. 4) and VVI (Fig. 5) respectively. Obviously, the annotations are new. For VDD, the atrial state machine A is that of AAI without the Pacing state and its arrows, and with two new

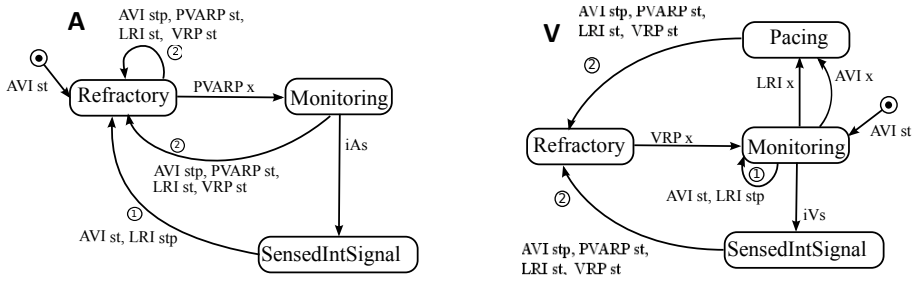

Fig. 11. VDD - atrium, ventricle processs

arrows into state Refractory. The ventricular state machine V is precisely that of VVI, with two extra arrows from state Monitoring. Secondly, for all modes involving A,V synchronisation, this is achieved by marking synchronising A,V event pairs (① and ② in Fig. 11). Each of these synchronisations takes the CSP [9] meaning of shared events in the two state machinees. That is, each instance of mark ② represents the *same* event, actioning a state transition simultaneously in each state machine depending on its current state.

The reader may wish to refer to timing diagram Fig. 3 and the time interval definitions of section 3, as well as Fig. 11 for this discussion. As before, in each channel we have an intrinsic-signal-sensed event and state, a pacing state, and internal pacemaker time interval events. We define both state machines to start at atrium intrinsic signal or pace, with the AVI[4] interval starting (AVI st). Thus A starts in Refractory state and V in Monitoring state.

A remains Refractory until either (i) synchronising transition ② occurs because of a ventricular paced or sensed event, or (ii) expiry of PVARP (PVARP x) - the latter case arises when A is Refractory after a ventricular event and PVARP has started. In this latter case A enters Monitoring, i.e. becomes enabled to sense any intrinsic atrial signal. When Monitoring, synchronising transition ② can occur because of a ventricular event. Otherwise, when sensing detects an intrinsic atrial signal (iAs, eg 2nd AS in Fig. 3), A moves to state SensedIntSignal, then on via starting AVI, and stopping LRI to Refractory. The latter transition signals this atrial event to V via synchronisation ①.

V starts in Monitoring state with AVI started. It can move to Pacing either (i) when AVI expires after an atrial signal, or (ii) when LRI expires (LRI x). The latter will happen, if after a ventricular signal or pace, no atrial signal is sensed within the LRI. From Monitoring, sensing in V may detect an intrinsic V-signal iVs (e.g. 1st VS in Fig. 3) and enter state SensedIntSignal. Both Pacing and SensedIntSignal states transition to Refractory, signalling the ventricular event to A via synchronisation ② (e.g. 1st VP, 1st VS resp. in Fig. 3): AVI stops, PVARP, VRP and LRI start. The remaining transition from Monitoring is an auto-transition, via synchronisation ① indicating an intrinsic atrial signal from A. Finally, if V is Refractory, on expiry of VRP it will go to Monitoring.

[4] In dual chamber modes the ARP is replaced by the AVI.

5.1 VDD - Event-B development

To develop an Event-B model of VVD mode we follow an approach very similar to what we have presented in Section 4.1. Our aim is to maintain simplicity and comprehensibility while achieving a high level of reusability. To realise these goals, we use solution state machine Fig. 6 and Event-B model Fig. 7 as templates and instantiate them by postfixing necessary elements with the name of channels A and V.

In the most abstract level to model VVD mode we use a single Event-B machine. However to achieve the above mentioned goals of comprehensibility and high level of reuse, events of each channel are kept separate from each other. Accordingly we use separate state variables to represent the state of each channel. Thus we use two state variables, namely $state\_a$ and $state\_v$, to represent the state of channels A and V respectively. To model A channel we have $Refractory\_A$ and $Monitor\_A$ events, acting on $state\_a$. Note that in VVD mode no atrial pacing is taking place. To model V channel we have $Refractory\_V$, $Monitor\_V$ and $Pace\_V$ events acting on $state\_v$. These are five events and two state variables in total to represent VDD mode. This abstract Event-B model is simple, so we skip further discussion of it and its corresponding state machine. Instead, we focus on the first refinement where we introduce the synchronisation between the two channels as well sensing of both channels and pacing of the V channel.

The first refinement is illustrated by the solution-space state machine of Fig. 12, and the corresponding Event-B model is presented in Fig. 13. This state machine corresponds to the problem state machine of Fig. 11. The synchronisations between the two channels are introduced in the form of a flag for each channel: the ventricle-atrium signal $v2a\_signal$, and the atrium-ventricle signal $a2v\_signal$. When a ventricular intrinsic signal, denoted by $SensedIntSignal\_V$ event, is detected or a ventricular pace, denoted by $Start\_Pacing\_V$ event, happens then the $v2a\_signal$ is set to $TRUE$. This in turn causes A channel to re-enter a refractory state by executing $Refractory2\_A$, which also resets the $v2a\_signal$ to $FALSE$. Similarly, sensing of an atrium intrinsic signal, denoted by $SensedIntSignal\_A$ event, sets the $a2v\_signal$ to $TRUE$. This will cause the ventricle channels to re-enter $StartMonitoring$ state by executing $Start\_Monitoring2\_V$, which also resets $a2v\_signal$ to $FALSE$.

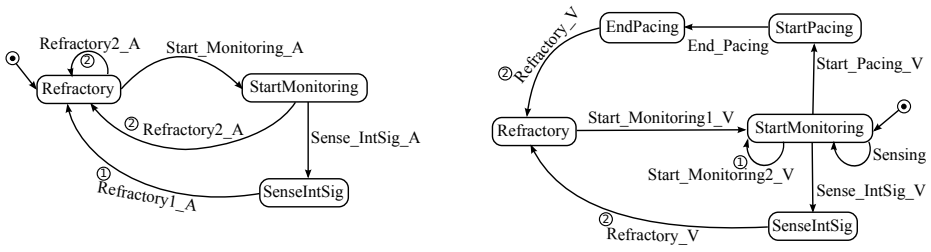

Fig. 12. VDD - solution space state machines

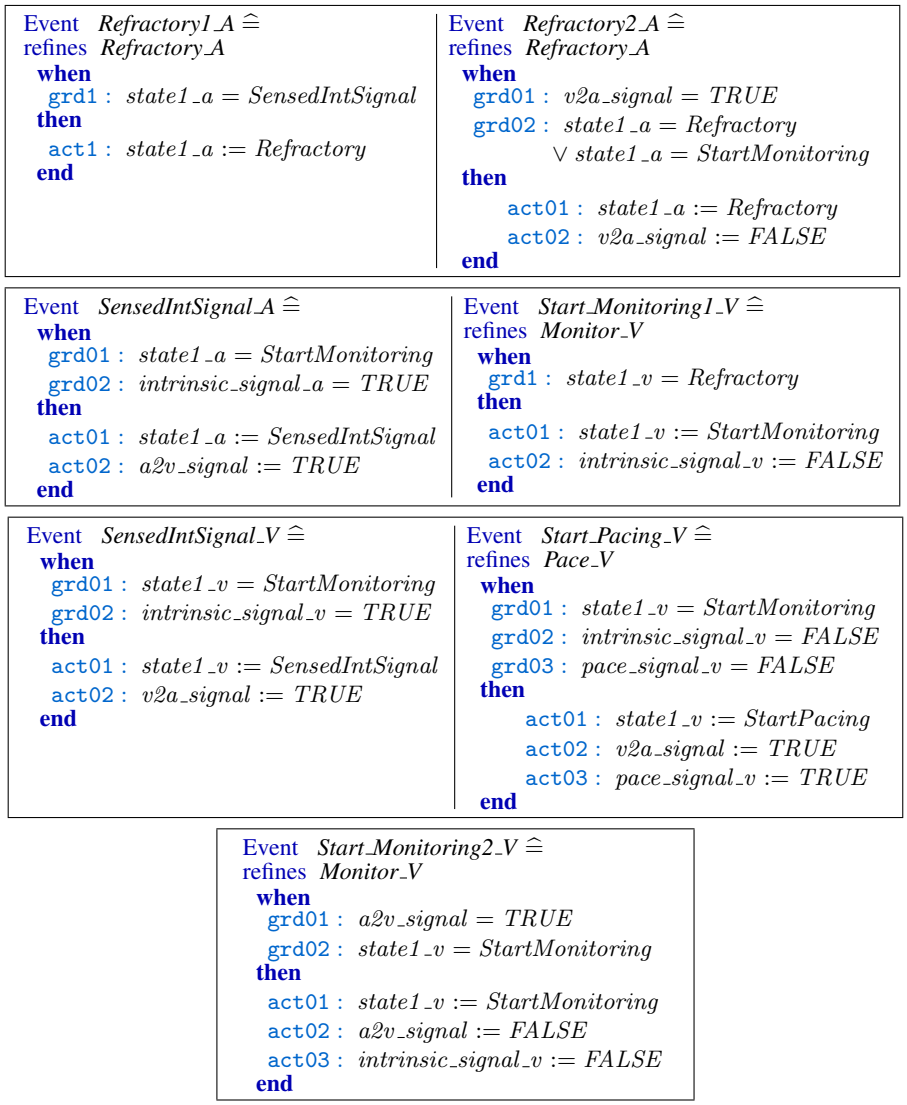

Fig. 13. First Refinement of VDD Mode: Introducing Synchronisation

In this refinement $Refractory1\_A$ and $Start\_Monitoring1\_V$ are ordinary refinements but $Refractory2\_A$ and $Start\_Monitoring2\_V$ are synchronisation events representing transitions of Fig. 12. Note that we have used ① and ② on the state machine of Fig. 12 to demonstrate how synchronisations of Fig. 11 are implemented.

In a second refinement of VDD mode, timing is introduced. This stage is very similar to the final part of Section 4.1 and we do not discuss it further here.

6 Conclusion and Related Work

This work is a methodological contribution to combining two interpretations of state machines in a reuse-intensive setting with demanding timing requirements. We have developed problem- and solution-domain models in state machines and Event-B for five modes of the pacemaker, reporting here on AAI, VVI and VDD. These parallel efforts have helped us to bridge the problem/requirements to solution/implementation space gap. We have shown how state machines act both as succinct accounts of observable pacemaker behaviour in terms of electrocardiology concepts, and as useful shorthands for Event-B developments. Since the problem-domain model is flat, only part of its information is used on the corresponding solution-domain model, which is then seen to be the first refinement of a 3-layered development.

The idea of two communicating machines for atrial and ventricular channels emerged naturally in the Event-B modelling, from reusing the AAI model in constructing VDD. Significant commonality in VDD and other dual-channel mode models emerged, as discussed in earlier sections. Thus, synchronising two-state-machine problem models were produced for VDD, DVI and DDD. The correspondence of the AAI problem and solution state machine models provided a guide for the construction of solution models for all modes. We have shown the design decision required to implement the ideal atrial-ventricular channel synchronisation in the solution model.

The expressive power of abstraction and refinement in Event-B has enabled us to give a simple state-based functional model, later applying current Event-B techniques to model more intricate timing aspects. This is an advantage of our approach in comparison to previous work. All proof obligations associated with the Event-B models have been discharged using Rodin built-in provers. Clearly there is further design, verification and refinement to be done, in particular aspects such as hysteresis and adaptive-rate pacing yet to be covered in further refinements.

For the pacemaker application with its many modes, the problem and solution models represent useful repositories of requirements and formal models, respectively. They are useful since the graphic notation is a shorthand, making the text of larger narrative and Event-B models respectively more comprehensible. Further work is required to define units of reuse in these models, and tool support to enable suitable refactoring and generic instantiation. Other possible avenues to investigate are graphical animation of the Event-B models using RODIN plugins, and the application of the RODIN state machine plugin to the solution models. The latter would be a good platform for further investigation of channel synchronisation design decisions, and ultimately, the mapping of such decisions to code.

References

[1] J.-R. Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[2] J. R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: An open toolset for modelling and reasoning in event-b. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(6):447–466, 2010.

[3] S.S. Barold, R.X. Stroobandt, and A.F. Sinnaeve. *Cardiac Pacemakers and Resynchronization Step-by-Step*. Wiley Blackwell, 2nd edition, 2010.

[4] Michael Butler and Jerome Falampin. An approach to modelling and refining timing properties in B. In *Refinement of Critical Systems (RCS)*, January 2002.

[5] D. Cansell, D. Méry, and J. Rehm. Time constraint patterns for event b development. In *B 2007*, volume 4355 of *LNCS*, pages 140–154. Springer, 2007.

[6] National Instruments Corporation. www.zone.ni.com.

[7] K.A. Ellenbogen and M.A. Wood. *Cardiac Pacing and ICDs*. Blackwell Science, 3rd edition, 2002.

[8] A.O. Gomes and M.V. Oliveira. Formal development of a cardiac pacemaker: From specification to code. In Jim Davies, Leila Silva, and Adenilso da Silva Simão, editors, *SBMF*, volume 6527 of *LNCS*, pages 210–225. Springer, 2010.

[9] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[10] Z. Jiang, M. Pajic, and R. Mangharam. Cyber-physical modeling of implantable cardiac medical devices. *Proceedings of the IEEE*, 100(1):122–137, 2012.

[11] Cliff B. Jones, Peter W. O'Hearn, and Jim Woodcock. Verified software: a grand challenge. *IEEE Computer*, 39(4):93–95, 2006.

[12] K.E. Klabunde. *Cardiovascular Physiology Concepts*. Lippincott Williams & Wilkins, 2nd edition, 2011.

[13] National Heart Lung and Blood Institute of the NIH. http://www.nhlbi.nih.gov/health/health-topics/topics/hhw/electrical.html, 2012.

[14] H.D. Macedo, P.G. Larsen, and J. Fitzgerald. Incremental development of a distributed real-time model of a cardiac pacing system using vdm. In *FM2008*, volume 5014 of *LNCS*. Springer, May 2008.

[15] D. Méry and N.K. Singh. Functional Behavior of a Cardiac Pacing System. *International Journal of Discrete Event Control Systems (IJDECS)*, December 2010.

[16] D. Méry and N.K. Singh. Trustable formal specification for software certification. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (2)*, volume 6416 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2010.

[17] C. Métayer, J.-R. Abrial, and L. Voisin. Event-B Language. Technical Report Deliverable 3.2, EU Project IST-511599 - RODIN, May 2005. http://rodin.cs.ncl.ac.uk.

[18] Boston Scientific. Pacemaker system specification. Technical report, 2007. http://sqrl.mcmaster.ca/pacemaker_spec.htm.

[19] C. Snook. Uml-B. http://wiki.event-b.org/index.php/UML-B, 2012.