ICSE 2009 SCORE PROJECT

A Simple Pacemaker Implementation

Valerio Panzica La Manna, Andrea Tommaso Bonanno, Alfredo Motta

Last revision: 02/28/2009

Abstract

In the context of real time and safety-critical systems the usage of formal methods is of paramount importance. The artificial pacemaker is a perfect example of a system whose behavior needs to be well and formally specified in order not to lead to possible mistakes. This paper shows how an extensive usage of formal methods written in the TRIO+ formal language can be applied to the specification of three different functioning modes of the pacemaker (AAT, VVI and DDD), and how the stated requirements can be validated through the use of the SAT checker called Zot and the specification and verification system called TVS (TRIO/PVS). Finally, we developed a Java software prototype that simulates the behavior of the pacemaker as described by our formal specification of the requirements.

1. Introduction

Boston Scientific has made available a natural language requirements document [3] for a previous generation pacemaker. This document has become the basis for a Grand Challenge proposed by McMaster University Software Quality Research Laboratory (Canada) [1], and is ideal for realistic student projects .

The Pacemaker developed in this project is a typical example of

- **Safety-critical system** because its failure or malfunction can compromise the health of the patient.
- **Real-time system** because its correct behavior strictly depends on timing constraints.

Formal models have a valuable role to play in validating requirements and designs for real-time safety-critical systems in early development stages. Rapid feedback from the analysis of such models has the potential to reduce the risk of expensive re-working as a consequence of the late-stage detection of defects and the generation of proofs that the code meets the requirements improves the software quality.

However, models that incorporate the description of functionality alongside timing behavior are themselves potentially complex. Moving too rapidly to such a complex model can increase modeling and design costs in the long run. In order to gain full value from formal

modeling and analysis, a systematic approach to constructing and validating models is required. This work is focused on the development of an industrial application with the use of formal modeling techniques that satisfy the requirements discussed above .

The approach proposed is based on modeling the system specification using the TRIO language [5] and exploiting its powerful tools to perform analyses of the system.
Using the resulting formal specifications, a simulator in Java was implemented in order to demonstrate that the final result completely reflects the expected behavior of the system.

2. Goal

The goal of the project is to build a simple but complete implementation of a pacemaker, targeting a subset of its functioning modes.
The choice to implement only these modes is due to the fact that each of them causes a totally different behavior of the heart that corresponds to different formal modeling and analysis of the system.
However, this prototype, is designed as a component that can be easily integrated with others in order to achieve a pacemaker that is able to perform many functionalities.

This software engineering project is developed following the well known Waterfall Software life-cycle :

- Requirements
- Design
- Implementation
- Testing

The approach proposed for the requirements consists of two further steps:

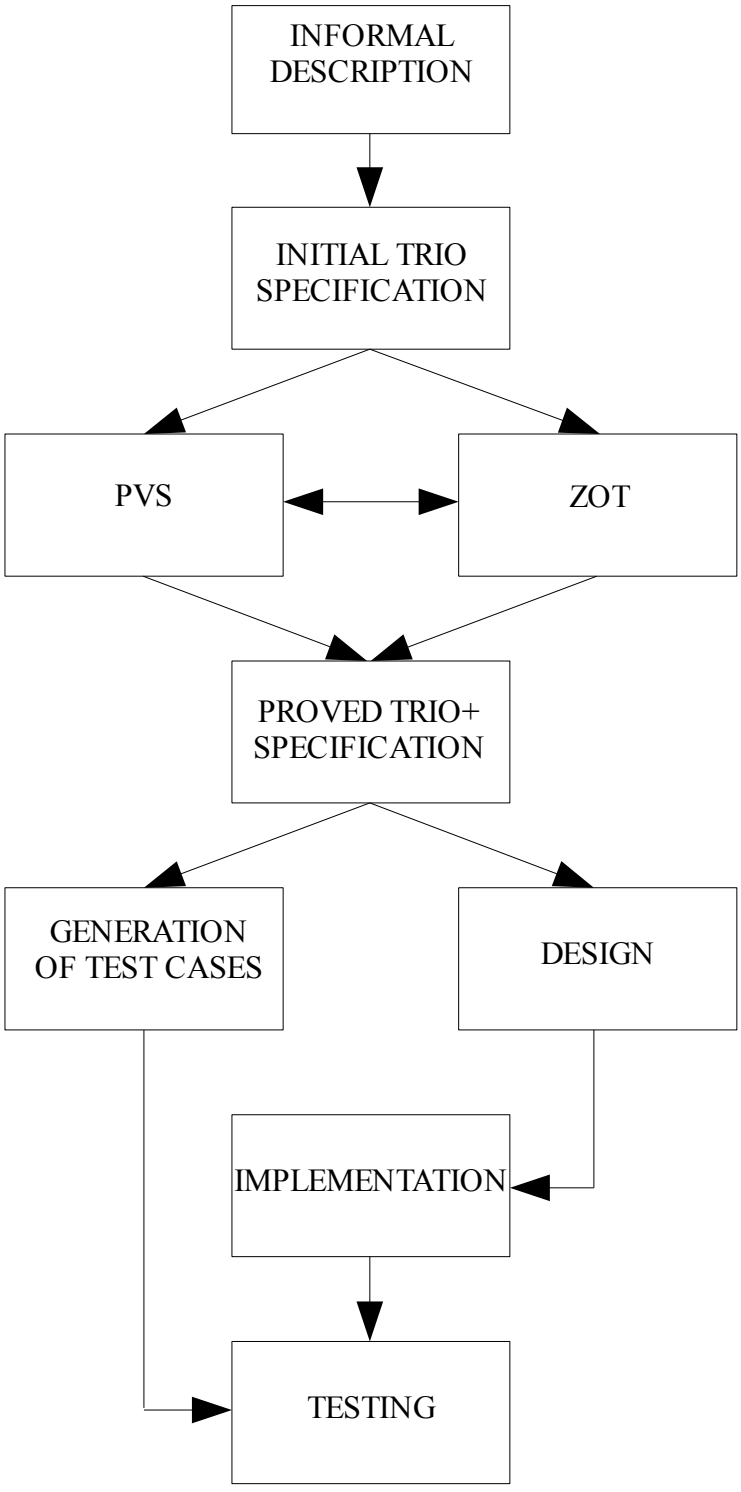
1) Defining a global specification of the system in the TRIO language (modeling)
2) Performing an iterative analytic process in which every TRIO axiom is added to the final specification only if:
    - The specification with this new axiom is satisfiable (ZOT SAT checker).
    - The new axiom is crucial for the PVS proof of some properties (no axiom redundancy).

ZOT was used to check the satisfiability of the overall axiomatization, PVS to prove the system properties.

For the purpose of this project we mainly focus on the requirements step for three main reasons:

- The final specification written in TRIO+ defines the components of the system and the interaction among them. So the TRIO+ specification corresponds to the **design** component view of the system.
- Each of this component is a set of axioms describing its behavior. For this reason the **implementation** step corresponds to the translation of these axioms, from TRIO language into Java.

- The set of axioms produced, beside its correctness, implies some extremely important properties: Utility and Security. After being formally proved, verifying these properties in the Java simulator is the goal of the **testing** step.


INFORMAL DESCRIPTION
INITIAL TRIO SPECIFICATION
PVS
ZOT
PROVED TRIO+ SPECIFICATION
GENERATION OF TEST CASES
DESIGN
IMPLEMENTATION
TESTING

3. Background

3.1 Heart

The heart is a muscular organ responsible for pumping blood through the blood vessels by repeated, rhythmic contractions. The heart has its own internal electrical system that controls the speed and rhythm of the heartbeat. With each heartbeat, an electrical signal spreads from the top of the heart to the bottom. As it travels, the electrical signal causes the heart to contract in an organized manner and pump blood.
A heartbeat is a single cycle in which the heart's chambers relax and contract to pump blood. This cycle includes the opening and closing of the two inlet and outlet valves of the right and left ventricles of the heart.

Each heartbeat has two basic parts: diastole, and atrial and ventricular systole. During diastole, the atria and ventricles of the heart relax and begin to fill with blood. At the end of diastole, the heart's atria contract (atrial systole), pumping blood into the ventricles, and then begin to relax. The heart's ventricles then contract (ventricular systole), pumping blood out of the heart.

Each beat of the heart is set in motion by an electrical signal from within the heart muscle. In a normal, healthy heart, each beat begins with a signal from the SA node. This is why the SA node is sometimes called the heart's natural pacemaker. The pulse, or heart rate, is the number of signals the SA node produces per minute.

3.2 Pacemaker

A pacemaker [8,9,10] is a small device that's placed under the skin of the chest or abdomen to help control abnormal heart rhythms. This device uses electrical pulses to prompt the heart to beat at a normal rate. Pacemakers are used to treat heart rhythms that are too slow, fast, or irregular. These abnormal heart rhythms are called arrhythmias. Pacemakers can relieve some symptoms related to arrhythmia, such as fatigue (tiredness) and fainting. A pacemaker can help a person who has an abnormal heart rhythm resume a more active lifestyle. Depending on which arrhythmias is present, modern pacemakers provides different functioning modes that perform different kinds of therapeutic behavior. In order to distinguish between different functioning modes, a code mechanism has been introduced. The code has evolved over several years to accommodate changes in pacing systems and there have been recommendations for up to a five-position code with multiple letters. From a practical point of view, the four-position code described in Table 1 represents general usage.

Position 1 refers to the chamber(s) being paced. V stands for ventricle, A stands for atrium, and D stands for dual (atrium and ventricle). There is really no O in this setting (an older implantable cardioverter defibrillator that did not have pacing backup could be designated O, but this is of historic interest only). Manufacturers will often designate S for single. This means it can be used to pace either the atrium or the ventricle. They are simply describing it more accurately and not designating it as one or the other since it can be used for either.

Position 2 refers to the chamber(s) being sensed. Again, V is for ventricle, A is for atrium, and D is for dual (atrium and ventricle). Again, the designation of S is often used by the manufacturer in a generic manner because of its potential application for either atrial or ventricular placement. In position 2, the designation O refers to absent sensing (and thus refers to fixed, asynchronous pacing). When a magnet is placed over most pacemakers, the sensing is disabled; for instance, a VVI pacemaker would become VOO.

Position 3 refers to the device's response to sensing. I represents the inhibited

mode, meaning that when the pacemaker senses an event, it will be inhibit pacing for that cycle; this is the most common form of sensing. T indicates a triggered response. When the pacemaker senses an event, it will trigger the device to deliver a pacing stimulus. In single-chamber situations, the sensed event and triggered impulse occur within the same chamber. When dual-chamber terminology is introduced, many new readers to pacemakers are confused by the third D. The third D refers to the ability to trigger a spike or to inhibit. In particular, an atrium event allows a triggered ventricular response in the ventricle. The main goal of this mode is to achieve what is called the AV synchrony between the two chambers. For example in VVI functioning mode there is no relation between atrium and ventricle events, meaning that when we provide a ventricular pulse this may occur at an arbitrary distance from an atrium event because we are simply ignoring what is happening in the other chamber. DDD mode instead uses atrium events to trigger ventricular pulses after a fixed AV delay without sensing any ventricular event.

Position 4 simply has an R added if the patient has rate modulation, in which a sensor is used to modify the heart rate of the pacemaker based on the patient's activity or metabolic need.

This is the functioning mode table from the Boston Scientific pacemaker specification. In this project we will concentrate on the VVI, DDD and AAT functioning modes, leaving rate modulation to future improvements.

	I	II	III	IV (optional)
Category	**Chambers Paced**	**Chambers Sensed**	**Response To Sensing**	**Rate Modulation**
Letters	O–None A–Atrium V–Ventricle D–Dual	O–None A–Atrium V–Ventricle D–Dual	O–None T–Triggered I–Inhibited D–Tracked	R–Rate Modulation

Table 1: Boston Scientific functioning modes

3.3 VVI

VVI mode is a single-chamber pacing mode. This means that this activity is concentrated only on one chamber of the heart, in this case the ventricle chamber. VVI pacing is most commonly used for patients with chronic atrial fibrillation and a slow ventricular response. Atrial fibrillation is a cardiac arrhythmia (abnormal heart rhythm) that involves the two upper chambers (atria) of the heart. A conclusive indication of atrial fibrillation is the absence of atrial events on an electrocardiogram (ECG). Pacing or sensing the atrium in these patients is meaningless because atrial events do not occur. VVI pacing could be used in a patient with such sinus syndrome as “backup pacing”, but during those times of pacing, AV synchrony would not be maintained. If the episodes of asystole are very rare and/or the patient is extremely inactive, this may not present a significant clinical problem. The addition of rate modulation (VVIR) is indicated when sinus node function is abnormal. Chronotropic incompetence is a common form of abnormal sinus node function in which appropriate increases in sinus rate do not occur. As a result the heart does not increase its own beat rate when it is needed by the patient.

3.4 AAT

The AAT mode acts only on the atrial chamber and its response to sensing is of a triggered kind. Its behavior is defined in Boston Scientific Specification [3] as:

“when a pulse is sensed in the atrium, a pulse is immediately triggered to the atrium itself; the pacemaker delivers stimuli to the atrium at a fixed rate in the absence of sensed atrial activity”. AAT is usually set as the pacemaker functioning mode when the patient has one of the following cardiac problems:

- Too weak atrial pulses: the atrial beats are too weak in terms of amplitude and/or width. The purpose of the pacemaker is to adjust both parameters of the beat, in order to resume an admissible state of the beat.
- Bradycardia: the patient has too few heart-beats per minute, or, in other words, he/she has a too slow heart rate. In particular the problem resides in the atrium, which may not pulse naturally and it needs an artificial help. The purpose of the pacemaker is to increase the heart rate in order to return the heart to an admissible and safe state.

Therefore the main challenges for a good implementation of this kind of pacemaker are: keeping the heart rate over the Lower Rate Limit(LRL) threshold while strengthening width and amplitude of pulses (Utility goal) and, in the meanwhile, maintaining a non-invasive behavior in the case the natural heart beating is still admissible (Safety goal).
Apart from the main parameters of the heart beat mentioned above, the main parameters the AAT mode needs to perform its actions are:

- Lower and Upper rate limits(LRL, URL): they are the lower and upper bounds for the heart rate in order to have an healthy and safe beating;
- Atrial Refractory Period (ARP): time interval following an atrial event during which time atrial events shall not inhibit nor trigger pacing;
- Post Ventricular Atrial Refractory Period: time interval following a ventricular event when an atrial cardiac event shall not inhibit an atrial pace nor trigger a ventricular pace.

3.5 DDD

DDD is another mode of the pulse generator (PG) treated in this project.

The acronym stands for:

D: both chambers paced.
D: both chambers sensed.
D: Tracked Response mode.

In the Tracked Response mode, an atrial sense shall cause a tracked ventricular pace after a programmed AV delay, unless a ventricular sense was detected beforehand.

DDD pacing is a form of dual-chambered pacing in which the atria and the ventricles are paced. In DDD pacing the atrium and the ventricle are sensed and paced or inhibited, depending on the native cardiac activity sensed. Other forms of dual-chambered pacing are available, such as DVI and VDD, but DDD is the most common. The principle advantage of dual-chambered pacing is that it preserves AV synchrony. Because of this advantage, dual-chambered pacing is increasingly common.

In DDD pacing, if the pacemaker does not sense any native atrial activity after a preset interval, it generates an atrial stimulus. An atrial stimulus, whether native or paced, initiates a period known as the AV interval. During the AV interval the atrial channel of the pacemaker is inactive, or refractory. At the end of the present AV interval, if no native ventricular activity is sensed by the ventricular channel, the pacemaker generates a ventricular stimulus. Following the AV interval, the atrial channel remains refractory during a short, post-ventricular atrial refractory period (PVARP) so as to prevent sensing the ventricular stimulus or resulting retrograde P waves as native atrial activity.

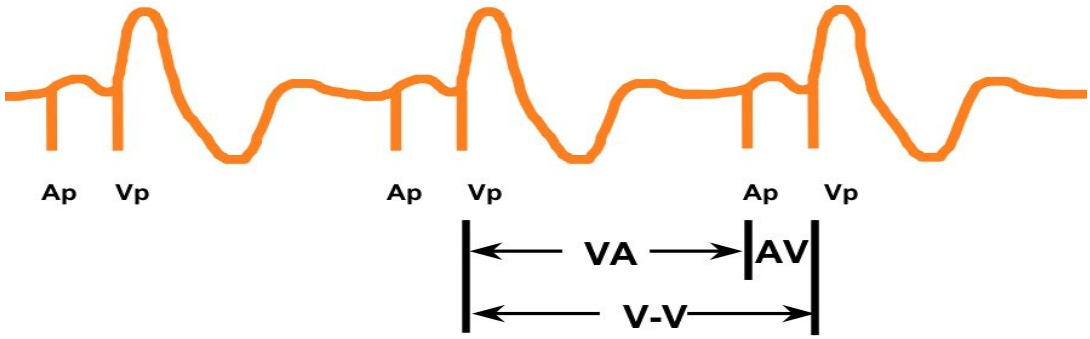


Figure 1: VA, AV, VV intervals (Ap stands for Atrial pulse, Vp for Ventricular pulse

3.6 Electrocardiogram

Electrocardiogram (ECG) is considered in this project as the main instrument used to show the natural patient heart rate (the input to the pacemaker) and the paced heart rate (the output). As the heart undergoes depolarization and repolarization, the electrical currents that are generated spread not only within the heart, but also throughout the body. This electrical activity generated by the heart can be measured by an array of electrodes placed on the body surface. The recorded tracing is called an electrocardiogram (ECG, or EKG). A "typical" ECG tracing is shown here in figure 2.

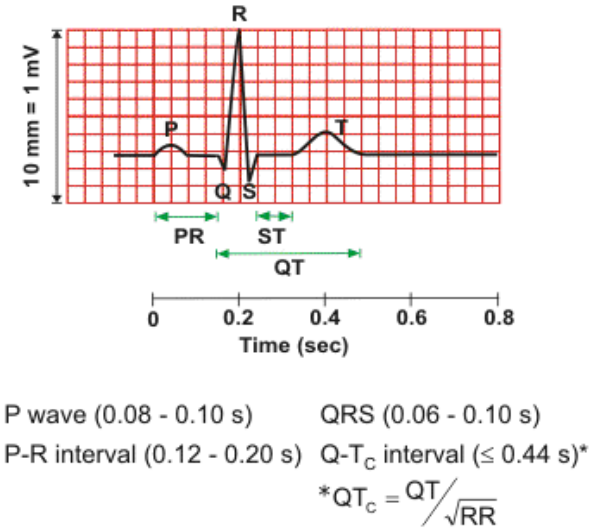


Figure 2: ECG Tracing

Artificial pacemakers use two electrical leads, placed in the atrium and ventricle chambers, to detect these signals. The different waves of the ECG represent the sequence of depolarization and repolarization of the atria and ventricles. Every different wave can be interpreted by the artificial pacemaker as an atrium event, or a ventricular event. The ECG is recorded at a speed of 25 mm/sec, and the voltages are calibrated so that 1 mV = 10 mm in the vertical direction. Therefore, each small 1-mm square represents 0.04 sec (40 msec) in time and 0.1 mV in voltage. Because the recording speed is standardized, one can calculate the heart rate from the intervals between different waves.

The P wave represents the wave of depolarization that spreads from the SA node

throughout the atria, and is usually 0.08 to 0.1 seconds (80-100 ms) in duration. The period of time from the onset of the P wave to the beginning of the QRS complex is termed the P-R interval, which normally ranges from 0.12 to 0.20 seconds in duration. This interval represents the time between the onset of atrial depolarization and the onset of ventricular depolarization. If the P-R interval is >0.2 sec, there is an AV conduction block, which is also termed a first-degree heart block if the impulse is still able to be conducted into the ventricles.

The QRS complex represents ventricular depolarization. Ventricular rate can be calculated by determining the time interval between QRS complexes. The duration of the QRS complex is normally 0.06 to 0.1 seconds. The shape of the QRS complex in the above figure is idealized. In fact, the shape changes depending on which recording electrodes are being used. The shape will also change when there is abnormal conduction of electrical impulses within the ventricles.

The T wave represents ventricular repolarization and is longer in duration than depolarization (i.e., conduction of the repolarization wave is slower than the wave of depolarization). T waves are a good example of noise for artificial pacemakers. T waves do not represents ventricular events; the ventricle is simply recharging itself to provide a new ventricular pulse. But T waves are not the only possible cause of noise. The signal detected by the artificial pacemaker may be full of random noise, even if the electrical lead is placed in contact with the heart chamber. Artificial pacemakers have to deal with this kind of problems and some solutions are in place thanks to some simple features. Lets consider the example of a T wave that may generate a false-positive for a QRS complex. There are two feature to avoid this problem.

Refractory periods: a QRS complex, a refractory period ignores all ventricular signals for a short time in order to avoid false-positives of ventricular events due to T waves or noise.

Sensing threshold of electrical signals; this is a minimum voltage that has to be read in order to detect an event of the heart. Because ventricular events (QRS complex). This higher voltage signals than do T waves, this can be recorded by the pacemaker as a threshold for ventricular events. This means that if a certain wave does not provide this minimum voltage this is not recorded as a ventricular event. The physician is responsible for this setting. A low threshold for heart events may cause false-positives, so the pacemaker probably will be inhibited. Alternatively, a high threshold may cause the pacemaker does not to sense heart events, so it probably will provide artificial pulses that are not needed by the heart. It is also interesting to observe how the ECG changes when the pacemaker is providing artificial pulses to the heart. It results into something unexpected that has to be taken in consideration. In the ECG that follows we can see a single chamber VVI pacemaker with a patient that suffers from atrial fibrillation (this is why we do not see any P-wave in this ECG).

The long spikes on the ECG represents the electrical pulses provided by the pacemaker, then a strange and wide QRS complex follows. What really matters for the pacemaker is how long this QRS complex is and what is the absolute value of his voltage in order to correctly guess refractory periods and thresholds for sensing.

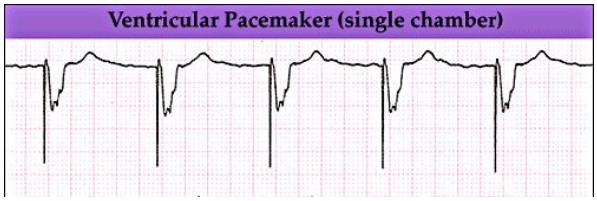


Figure 3: VVI pacemaker with patient suffering from atrial fibrillation

4. TRIO Specification

TRIO [4 and 5] (Tempo Reale ImplicitO) is a formal language and a method for the specification, analysis and verification of critical, real-time systems. The TRIO language is based on a metric extension of first-order temporal logic and exploits typical object-oriented features to support the managing of large, complex, and maintainable specifications. TRIO specifications are based on the definition of the Dist operator. Dist(A, t) is a TRIO formula and intuitively means that it holds if and only if A holds at distance t from the current instant (i.e., the instant when Dist(A, t) is evaluated) in the temporal domain.

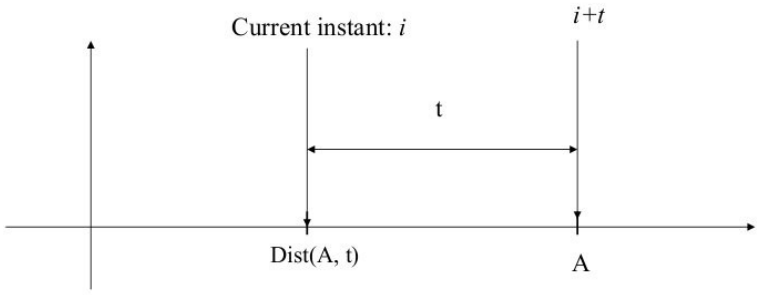


Figure 4: The Dist TRIO Operator

From this basic TRIO formula, it is possible to derive many different operators whose behavior is explained by their names. We have also added some explanatory text for each operator. All the operators are evaluated at the current time instant.

Futr(F, d) $\Leftrightarrow d \geq 0 \wedge$ Dist(F, d)
F will be true after a time interval of d.
Past(F, d) $\Leftrightarrow d \geq 0 \wedge$ Dist(F, -d)
F was true d time instants in the past.
Lastsee(F, d) $\Leftrightarrow \forall d' (0<d'<d \rightarrow$ Dist(F, d'))
F holds over a period of length d (boundaries of the interval are excluded).
Lastsie(F, d) $\Leftrightarrow \forall d' (0 \leq d'<d \rightarrow$ Dist(F, d'))
F holds over a period of length d (now is included, now +d is excluded).
Lastsei(F, d) $\Leftrightarrow \forall d' (0<d' \leq d \rightarrow$ Dist(F, d'))
F holds over a period of length d (now is excluded, now +d is included).
Lastsii(F, d) $\Leftrightarrow \forall d' (0 \leq d' \leq d \rightarrow$ Dist(F, d'))
F holds over a period of length d (boundaries of the interval are included).
Lastedee(F, d) $\Leftrightarrow \forall d'(0 < d'<d \rightarrow$ Dist(F, -d'))
F held over a period of length d in the past (boundaries of the interval are excluded).
Lastedie(F, d) $\Leftrightarrow \forall d'(0 \leq d'<d \rightarrow$ Dist(F, -d'))
F held over a period of length d in the past (now is included, now -d is excluded).
Lastedei(F, d) $\Leftrightarrow \forall d'(0 < d' \leq d \rightarrow$ Dist(F, -d'))
F held over a period of length d in the past (now is excluded, now -d is included).
Lastedii(F, d) $\Leftrightarrow \forall d'(0 \leq d' \leq d \rightarrow$ Dist(F, -d'))
F held over a period of length d in the past (boundaries of the interval are included).
Alw(F) $\Leftrightarrow \forall d$ (Dist(F, d))
F always holds.
SomF (A) $\Leftrightarrow \exists d(d \geq 0 \wedge$ Dist(F, d))
Sometimes in the future F will hold.
Withinee(F, d) $\Leftrightarrow \exists d'(0<d'<d \wedge$ Dist(F, d'))
F will occur within d time units.
Withinie(F, d) $\Leftrightarrow \exists d'(0 \leq d'<d \wedge$ Dist(F, d'))
F will occur within d time units.
Withinei(F, d) $\Leftrightarrow \exists d'(0<d' \leq d \wedge$ Dist(F, d'))
F will occur within d time units.
Withinii(F, d) $\Leftrightarrow \exists d'(0 \leq d' \leq d \wedge$ Dist(F, d'))
F will occur within d time units.
UpToNow (F) $\Leftrightarrow \exists d$ ($d > 0 \wedge$ Past (F, d) $\wedge$ Lastedee (F, d))
F held for a nonnull time interval that ended at the current instant.
Becomes (F) $\Leftrightarrow F \wedge$ UpToNow ($\sim$F)
F holds at the current instant but it did not hold for a nonnull interval that preceded the current instant.
NextTime (F, t) $\Leftrightarrow$ Futr (F, t) $\wedge$ Lastsie (F, t)
The first time in the future when F will hold is t time units apart from the current instant.
Until (A1, A2) $\Leftrightarrow \exists t$ ($t > 0 \wedge$ Futr (A2, t) $\wedge$ Lastsee (A1, t))
A1 holds until A2 becomes true.
Since (A1, A2) $\Leftrightarrow \exists t$ ($t > 0 \wedge$ Past (A2, t) $\wedge$ Lastedee (A1, t))
A1 held since A2 became true.

4.1 TRIO+ modular specification

The pacemaker system specification has been written using an extension of TRIO, called TRIO+ that uses the concept of class in order to group together sets of axioms that refer to the same component of the system. The following illustration shows an overview of the system and the connections between the components.
As you can see the whole system is composed by the Heart and the pacemaker.
The heart is a very simple model of how the heart behaves in response to artificial pulses

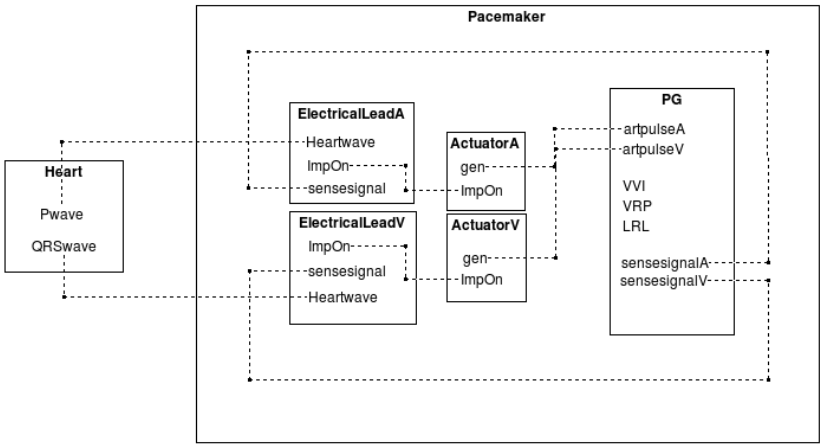


Figure 5: Component View

and natural pulses. It provides P waves and QRS waves with a certain heart rate in absence of artificial pulses. When an artificial pulse is generated, provided the heart reacts with a P wave or a QRS wave depending on the chamber in which the pulse has been generated.

The pacemaker itself has three main components:

Electrical leads: the leads that are physically connected to the heart chamber. They sense electrical signals from the heart. They are also responsible for providing artificial pulses. When a signal is sensed then *sensesignal* information is transmitted instantaneously to the PG.

Actuators: The electrical circuits responsible for providing the necessary difference in electrical potential for artificial pulses for a certain amount of time. When *gen* is true an artificial pulse is generated so *ImpOn* will be true for a certain amount of time.

PG: The *pulse generator*, which is responsible for implementing the therapeutic behavior of the pacemaker. It records all the information about functioning modes, refractory periods and thresholds. It also stores some fundamental parameters like the *lower rate limit* which is the minimum rate the pacemaker must guarantee for the patient's heart. Finally when an artificial pulse is needed, *artpulseA* or *artpulseV* will be true, so thanks to the connection to the *actuator*, a *gen* event will be true and, as a consequence, an electrical pulse will be generated for a certain amount of time.

4.2 PVS theorem prover

PVS [6] is a verification system: that is, a specification language integrated with support tools and a theorem prover. It is intended to capture the state-of-the-art in mechanized

formal methods and to be sufficiently rugged that it can be used for significant applications. PVS is a research prototype: it evolves and improves as we develop or apply new capabilities, and as the stress of real use exposes new requirements. A tool called TVS (TRIO/PVS) for translating TRIO+ specifications into PVS has been developed in *Politecnico di Milano*. Thanks to this feature, the pacemaker system specification has been imported into PVS in order to prove some fundamental properties of the system.

4.2.1 TVS system specification

The system is described by four sets of axioms, one for each component of the system. The axioms are then grouped together in the system class which states some conjectures about the system that must be proved. The heart class states the behavior of the heart in responses to atrial and ventricular contraction that can be provided by the heart itself or by the pacemaker. For the purpose of this project it is enough to consider the patient heart rate=PR, the time interval between two atrial pulse RR = (1/PR), the atrium-ventricle time interval=HAV, the post ventricular-atrial interval=HPV and the duration of the heart pulses.
RR= HAV+HPV.
These are the primary parameters that contribute to the heart behavior, and are used in the axioms below.
Electrical leads simply read electrical signals provided by the heart and by Actuators. The resulting signals are transmitted to the pacemaker thanks to the connection 3 and 4 in the system class (*pacemaker_class*). Actuators receive artificial pulse commands from the PG (connection 5 and 6 of the system class) and provide artificial pulses that have a duration PULSEDUR. PG states all the axioms abouts VVI mode and its main parameters which are the lower rate limit=LRL=1/TIMEOUT, and the ventricular refractory period=VRP. A good example to comment on is the *ignoresignalv* axiom of the PG class which states that we are receiving a ventricular event (*senseV*) if and only if we are sensing the signal from the electrical leads (*sensesignalV*) and we are not ignoring it (*ignoresignalV*).
It is also interesting to make some comments on the conjectures that have been stated in the system class (*pacemaker_class*). *Artpulsev* conjecture states that if the pacemaker provides an artificial pulse, then immediately it will sense an electrical signal as a response. *Natpulsev* conjecture states the same idea for the Heart, meaning that for every natural pulse an electrical signal is immediately generated. *Sensev1* conjecture states that our axiomatization of ventricular event (*senseV*) corresponds to the first time instant in which we sense an electrical signal from the ventricle chamber (QRS wave). *Det1* conjecture states that our system sense electrical signal is provided only by the heart and the pacemaker. *Utility* and *security* will be discussed in detail in the next chapter. Finally the *restore* properties is a variation of the utility property that focuses on how the pacemaker measures the heart rate. All these conjectures have been proved in PVS.

```
pacemakerHeart_class [ instances : TYPE+ , RR:posreal, HAV:posreal, HPV:posreal, PWAVEDUR:posreal, QRSWAVEDUR:posreal] :
THEORY
 BEGIN
  IMPORTING trio_base, states_and_events, trio_parametric_base[posreal]

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %% declarations
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   Pwave:     [instances ->State]
   QRSwave:   [instances ->State]
   natpulseA: [instances -> Event]
   natpulseV: [instances -> Event]
   sensesignalA: [instances -> Event]
   sensesignalV: [instances -> Event]
   senseA: [instances -> Event]
   senseV: [instances -> Event]
   ci : VAR instances

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %% axioms
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    pwave1: AXIOM Alw( Lasted_ie( NOT senseV(ci) AND NOT senseA(ci),HPV ) IFF natpulseA(ci))
    pwave2: AXIOM Alw( natpulseA(ci) AND UpToNow(NOT Pwave(ci)) IMPLIES Pwave(ci) )
    pwave3: AXIOM Alw(NOT natpulseA(ci) AND UpToNow(NOT Pwave(ci)) IMPLIES NOT Pwave(ci))
    pwave4: AXIOM Alw( Pwave(ci) IFF
                                        EX!(t1:posreal | t1<=PWAVEDUR) :
                                                Lasted_ii(Pwave(ci),t1) AND
                                                Lasts_ie(Pwave(ci), PWAVEDUR - t1) AND
                                                Past(UpToNow(NOT Pwave(ci)),t1) AND
                                                Futr(NOT Pwave(ci), PWAVEDUR - t1)
                                        )
    qrswave1: AXIOM Alw( Lasted_ie( NOT senseV(ci), HPV+HAV ) IFF natpulseV(ci))
    qrswave2: AXIOM Alw( natpulseV(ci) AND UpToNow(NOT QRSwave(ci)) IMPLIES QRSwave(ci))
    qrswave3: AXIOM Alw( NOT natpulseV(ci) AND UpToNow(NOT QRSwave(ci)) IMPLIES NOT QRSwave(ci))
    qrswave4: AXIOM Alw( QRSwave(ci) IFF
                                        EX!(t1:posreal | t1<=QRSWAVEDUR) :
                                                Lasted_ii(QRSwave(ci),t1) AND
                                                Lasts_ie(QRSwave(ci), QRSWAVEDUR - t1) AND
                                                Past(UpToNow(NOT QRSwave(ci)),t1) AND
                                                Futr(NOT QRSwave(ci), QRSWAVEDUR - t1)
                                        )

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% conjectures
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %from pwave3
    pwavecj: CONJECTURE Alw(Pwave(ci) AND UpToNow(NOT Pwave(ci)) IMPLIES natpulseA(ci))
    %from qrswave3
 qrswavecj:CONJECTURE Alw(QRSwave(ci) AND UpToNow(NOT QRSwave(ci)) IMPLIES natpulseV(ci))

END pacemakerHeart_class

pacemakerElectricalLead_class [ instances : TYPE+ ] : THEORY
 BEGIN
   IMPORTING trio_base, states_and_events, trio_parametric_base[posreal]

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %declarations
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ImpOn:     [instances -> State]
    Heartwave:  [instances -> State]
    sense: [instances -> Event]
    ci : VAR instances

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %axioms
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   lead1: AXIOM Alw( ImpOn(ci) OR Heartwave(ci) IFF sense(ci))

END pacemakerElectricalLead_class

pacemakerActuator_class [ instances : TYPE+, PULSEDUR:posreal ] : THEORY
   BEGIN
    IMPORTING trio_base, states_and_events, trio_parametric_base[posreal]
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %declarations
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    gen:    [instances -> Event]
    ImpOn:  [instances -> State]
    ImpOff: [instances -> State]
    ci : VAR instances

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %axioms
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    act1: AXIOM Alw( gen(ci) AND UpToNow(NOT ImpOn(ci)) IMPLIES ImpOn(ci) )
    act2: AXIOM Alw( NOT gen(ci) AND UpToNow( NOT ImpOn(ci)) IMPLIES NOT ImpOn(ci))
    act3: AXIOM Alw( ImpOn(ci) IFF
                                    EX!(t1:posreal | t1<=PULSEDUR):
                                                Lasted_ii(ImpOn(ci),t1) AND
                                                Lasts_ie(ImpOn(ci),PULSEDUR-t1) AND
                                                Past(UpToNow(NOT ImpOn(ci)),t1) AND
                                                Futr(NOT ImpOn(ci),PULSEDUR-t1))

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %conjectures
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %act2 is equivalent to actcj
    actcj: CONJECTURE Alw(UpToNow(NOT ImpOn(ci)) AND ImpOn(ci) IMPLIES gen(ci))
```

END pacemakerActuator_class

```
pacemakerPG_class [ instances : TYPE+, TIMEOUT:posreal,AV:posreal,VRP:posreal,ARP:posreal,PVARP:posreal ] : THEORY

 BEGIN

  IMPORTING trio_base, states_and_events, trio_parametric_base[posreal]

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %% declarations
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  HR: [instances -> TD_Term[posreal]]

  sensesignalV: [instances ->Event]
  sensesignalA: [instances ->Event]

  ignoresignalV: [instances ->State]
  ignoresignalA_ARP: [instances ->State]
  ignoresignalA_PVARP: [instances ->State]

  senseV: [instances ->Event]
  senseA: [instances ->Event]

  artpulseV: [instances -> Event]
  artpulseA: [instances -> Event]

  VVI:  [instances -> State]
  AAT:  [instances -> State]
  DDD:  [instances -> State]
  DDDR: [instances -> State]

  ci : VAR instances


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %% axioms
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %we can only perform artificial pulses if one of the functioning mode is on
  artpulse1: AXIOM Alw( artpulseA(ci) IMPLIES DDD(ci) OR AAT(ci) OR DDDR(ci) )
  artpulse2: AXIOM Alw( artpulseV(ci) IMPLIES DDD(ci) OR VVI(ci) OR DDDR(ci) )

  %functioning mode are mutually exclusive
  mutual: AXIOM Alw( (DDD(ci) IFF not ( VVI(ci) OR AAT(ci) OR DDDR(ci)))
                AND  (VVI(ci) IFF not ( DDD(ci) OR AAT(ci) OR DDDR(ci)))
                AND  (AAT(ci) IFF not ( VVI(ci) OR DDD(ci) OR DDDR(ci)))
                AND (DDDR(ci) IFF not ( VVI(ci) OR AAT(ci) OR DDD(ci))))

  %update heart rate
  updatehr: AXIOM FORALL(tt:Time),(t1:Time): (senseV(ci) AND Lasts(NOT senseV(ci),t1) AND Futr(senseV(ci),t1)
                 IMPLIES Futr(HR(ci)==LV(t1),t1))(tt)

  %heart rate updated only on heart contraction
  keephr:   AXIOM FORALL(tt:Time),(x1:posreal),(x2:Time):(HR(ci)==LV(x1) AND Lasts_ei(NOT senseV(ci),x2)
                 IMPLIES Lasts_ei(HR(ci)=LV(x1),x2))(tt)

  %In DDD an atrial sense shall cause a tracked ventricular pace after a programmed AV delay
  %unless a ventricular sense was detected beforehand.
  ax_ddd1: AXIOM Alw( DDD(ci) IMPLIES
                (artpulseV(ci) IFF Past(senseA(ci),AV) AND Lasted(NOT senseV(ci),AV)))
  ax_ddd2: AXIOM Alw( DDD(ci) IMPLIES
                 (Lasted(NOT senseA(ci),TIMEOUT) IFF artpulseA(ci)))

  %In AAT an atrial sense shall trigger an atrial artificial pulse immediatly
  ax_aat: AXIOM Alw( AAT(ci) AND senseA(ci) IMPLIES artpulseA(ci) )

  %In VVI a ventricular sense inhibits ventricular artificial pulses for Timeout
  ax_vvi: AXIOM Alw(VVI(ci) IMPLIES
                (Lasted(NOT senseV(ci),TIMEOUT) IFF artpulseV(ci)) )

  %The Ventricular Refractory Period shall be the programmed time interval fol-
  %lowing a ventricular event during which time ventricular senses shall not inhibit
  %nor trigger pacing.
  vrp1:  AXIOM Alw( ignoresignalV(ci) IFF
                  EX!(t1:posreal | t1<=VRP):
                          Lasted_ie(ignoresignalV(ci),t1) AND
                          Lasts_ie(ignoresignalV(ci),VRP-t1) AND
                          Past(senseV(ci),t1) AND
                          Futr(NOT ignoresignalV(ci),VRP-t1))
```

```
    vrp2: AXIOM Alw( senseV(ci) IMPLIES NowOn(ignoresignalV(ci)) )

    %the ARP shall be the programmed time interval following an atrial event during which time atrial
    %events shall not inhibit nor trigger pacing.
    arp1: AXIOM Alw( ignoresignalA_ARP(ci) IFF
                        EX!(t1:posreal | t1<=ARP):
                                            Lasted_ie(ignoresignalA_ARP(ci),t1) AND
                                            Lasts_ie(ignoresignalA_ARP(ci),ARP-t1) AND
                                            Past(senseA(ci),t1) AND
                                            Futr(NOT ignoresignalA_ARP(ci),ARP-t1))

    arp2: AXIOM Alw( senseA(ci) IMPLIES NowOn(ignoresignalA_ARP(ci)))

    %The PVARP shall be the programmable time interval following a ventricular event
    %when an atrial cardiac event shall not 1. Inhibit an atrial pace. 2. Trigger a ventricular pace.
    pvarp1: AXIOM Alw( ignoresignalA_PVARP(ci) IFF
                          EX!(t1:posreal | t1<=PVARP):
                                            Lasted_ie(ignoresignalA_PVARP(ci),t1) AND
                                            Lasts_ie(ignoresignalA_PVARP(ci),PVARP-t1) AND
                                            Past(senseV(ci),t1) AND
                                            Futr(NOT ignoresignalA_PVARP(ci),PVARP-t1))

    pvarp2: AXIOM Alw( senseV(ci) IMPLIES NowOn(ignoresignalA_PVARP(ci)))

    %When ignoresignal is true we must ignore signal from electrical leads
    ignoresignala: AXIOM Alw( sensesignalA(ci) AND NOT ignoresignalA_ARP(ci) AND NOT ignoresignalA_PVARP(ci)
                        IFF senseA(ci) )
    ignoresignalv: AXIOM Alw( sensesignalV(ci) AND NOT ignoresignalV(ci) IFF senseV(ci) )



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %conjectures
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %from ignoresignalv
    sensevcj: AXIOM Alw( senseV(ci) IMPLIES sensesignalV(ci))



END pacemakerPG_class

pacemaker_class [ instances : TYPE+ ] : THEORY


  BEGIN

    IMPORTING trio_base, states_and_events, trio_parametric_base[posreal]

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% system parameters
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    TIMEOUT:posreal
    PULSEDUR:posreal=0.4
    AV:posreal=150
    VRP:posreal=320
    ARP:posreal=250
    PVARP:posreal=250

    HAV:posreal
    HPV:posreal
    PWAVEDUR: posreal=0.2
    QRSWAVEDUR: posreal=0.4
    RR:posreal=HAV+HPV

    EPS:posreal


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% import classes
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    PG_type : TYPE+
    IMPORTING pacemakerPG_class[[instances, PG_type],TIMEOUT,AV,VRP,ARP,PVARP] AS PG
    PG : PG_type

    Heart_type : TYPE+
    IMPORTING pacemakerHeart_class[[instances, Heart_type],RR,HAV,HPV,PWAVEDUR,QRSWAVEDUR] AS Heart
    Heart : Heart_type
```

```
Actuator_type : TYPE+
IMPORTING pacemakerActuator_class[[instances, Actuator_type],PULSEDUR] AS Actuator
actuatorV : Actuator_type
actuatorA : Actuator_type

ElectricalLead_type : TYPE+
IMPORTING pacemakerElectricalLead_class[[instances, ElectricalLead_type]] AS ElectricalLead
elV : ElectricalLead_type
elA : ElectricalLead_type

ci : VAR instances



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% connections
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

conn1: AXIOM Heartwave(ci,elV)=QRSwave(ci,Heart)
conn2: AXIOM Heartwave(ci,elA)=Pwave(ci,Heart)

conn3: AXIOM sense(ci,elV)=sensesignalV(ci,PG)
conn4: AXIOM sense(ci,elA)=sensesignalA(ci,PG)

conn5: AXIOM artpulseV(ci,PG)=gen(ci,actuatorV)
conn6: AXIOM artpulseA(ci,PG)=gen(ci,actuatorA)

conn7: AXIOM ImpOn(ci,actuatorV)=ImpOn(ci,elV)
conn8: AXIOM ImpOn(ci,actuatorA)=ImpOn(ci,elA)

conn9:  AXIOM sense(ci,elV)=sensesignalV(ci,Heart)
conn10: AXIOM sense(ci,elA)=sensesignalA(ci,Heart)

conn11: AXIOM senseA(ci,PG)=senseA(ci,Heart)
conn12: AXIOM senseV(ci,PG)=senseV(ci,Heart)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% axioms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c_timeout: AXIOM TIMEOUT>VRP+EPS
c_rr:      AXIOM RR>VRP+EPS

%If we sense a pulse now, we are sure that in the future there will be another pulse
simple1:   AXIOM Alw( senseV(ci,PG)
                      IMPLIES EX!(x1:postime): Lasts(NOT senseV(ci,PG),x1) AND Futr(senseV(ci,PG),x1) )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% conjecture
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%artificial pulses are sensed immediatly
artpulsev: AXIOM Alw( artpulseV(ci,PG) AND VVI(ci,PG) IMPLIES senseV(ci,PG) )


%natural pulses are sensed immediatly
natpulsev: AXIOM Alw( natpulseV(ci,Heart) IMPLIES senseV(ci,PG) )


%sensev represent always the first instant of the signal
sensev1: AXIOM Alw( senseV(ci,PG) IMPLIES UpToNow(NOT sensesignalV(ci,PG)) )


%senseV only if natpulsev or artificial pulse v
det1: AXIOM Alw( senseV(ci,PG) IMPLIES natpulseV(ci,Heart) OR artpulseV(ci,PG) )


%We do something useful when RR>Timeout
utility1: CONJECTURE Alw( senseV(ci,PG) AND RR>TIMEOUT
                          AND VVI(ci,PG) AND Lasts_ii(VVI(ci,PG), TIMEOUT)
                          IMPLIES Futr(senseV(ci,PG), TIMEOUT) AND Lasts(NOT senseV(ci,PG), TIMEOUT))

%We do not behave wrongly when RR<TIMEOUT
security1:CONJECTURE Alw( senseV(ci,PG) AND RR<TIMEOUT
                          AND VVI(ci,PG) AND Lasts_ii(VVI(ci,PG), RR)
                          IMPLIES Futr(senseV(ci,PG), RR) AND Lasts(NOT senseV(ci,PG), RR))

%If our computation of HR is less than RR then after Timeout we are sure that it will be over Timeout
```

```
  restore: CONJECTURE Alw( EX!(x1:posreal|x1<=RR):(HR(ci,PG)==LV(x1)) AND RR>TIMEOUT
                           AND VVI(ci,PG) AND Lasts_ii(VVI(ci,PG),2*TIMEOUT)
                           IMPLIES Futr(HR(ci,PG)==LV(TIMEOUT), 2*TIMEOUT ))

END pacemaker_class
```

4.2.2 Utility and Security proof

At this point we can examine what a possible proof [11] of the specification looks like. What we are mainly interested in, are the *utility* and *security* properties of the system. You may have noticed that these properties have been stated in the conjectures section of the system class (*pacemaker_class*). The utility property states that when the heart rate is lower than the objective rate LRL, the pacemaker has to provide artificial pulses in order to increase the effective heart-beat rate. Reasoning in time, this means that if RR=1/HeartNaturalRate > TIMEOUT=1/LRL and if we sense a ventricular contraction in the current time instant, the next ventricular contraction will be after a time interval of TIMEOUT, moreover during this time interval we will not sense any other ventricular contraction because neither the heart or the pacemaker will provide any electrical pulse.
The safety property is the opposite one, means that the pacemaker will not provide artificial pulses when the heart rate is greater than the objective rate LRL. Again, reasoning in time this means that if RR=1/HeartNaturalRate < TIMEOUT=1/LRL and if we sense a ventricular contraction in the current time instant, the next ventricular contraction will be after a time interval of RR, moreover during this time interval we will not sense any other ventricular contraction because neither the heart nor the pacemaker will provide any electrical pulse. For the purpose of this document only the utility proof is reported.

```
%The proof has mainly two goals, so the proof tree has a big split in the first passages
utility1 :
{-1}  senseV(ci!1, PG)(tt!1)
{-2}  RR > TIMEOUT
{-3}  VVI(ci!1, PG)(tt!1)
{-4}  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
{1}   senseV(ci!1, PG)(tt!1 + TIMEOUT) AND
      Lasts(NOT senseV(ci!1, PG), TIMEOUT)(tt!1)

%utility 1.1 refers to the half tree of the proof, the one that proves that it is true that we have a
ventricular contraction at tt!1+TIMEOUT
utility1.1 :
{-1}  FORALL (ci: instances):
       Alw(senseV(ci, PG) IMPLIES
           EX! (x1: postime):
             Lasts(NOT senseV(ci, PG), x1) AND Futr(senseV(ci, PG), x1))
[-2]  senseV(ci!1, PG)(tt!1)
[-3]  RR > TIMEOUT
[-4]  VVI(ci!1, PG)(tt!1)
[-5]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   senseV(ci!1, PG)(tt!1 + TIMEOUT)

%...Omitted passages...%
%We have sensed a ventricular contraction at tt!1, we know that there will be another ventricular
contraction at tt!1+x!1. We need to quantify x!1 to complete the proof. We will prove that x!1=TIMEOUT
utility1.1 :

{-1}  Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
{-2}  senseV(ci!1, PG)(x!1 + tt!1)
[-3]  senseV(ci!1, PG)(tt!1)
[-4]  RR > TIMEOUT
[-5]  VVI(ci!1, PG)(tt!1)
[-6]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
```

```
[1]   senseV(ci!1, PG)(TIMEOUT + tt!1)
```

%To prove that x!1=TIMEOUT we will consider both cases x!1>TIMEOUT and x!1<TIMEOUT and we will prove that this will throw two contradiction.

```
Rerunning step: (case "x!1>TIMEOUT")
Case splitting on
   x!1 > TIMEOUT,
this yields  2 subgoals:
utility1.1.1 :

{-1}  x!1 > TIMEOUT
[-2]  Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-3]  senseV(ci!1, PG)(x!1 + tt!1)
[-4]  senseV(ci!1, PG)(tt!1)
[-5]  RR > TIMEOUT
[-6]  VVI(ci!1, PG)(tt!1)
[-7]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   senseV(ci!1, PG)(TIMEOUT + tt!1)
```

%We use the VVI main axiom to introduce a new fact

```
Rerunning step: (lemma ax_vvi)
Using instance
  pacemakerPG_class
     [[instances, PG_type], TIMEOUT, AV, VRP, ARP, PVARP].ax_vvi
Applying ax_vvi
this simplifies to:
utility1.1.1 :

{-1}  FORALL (ci: [instances, PG_type]):
       Alw(VVI(ci) IMPLIES
            (Lasted(NOT senseV(ci), TIMEOUT) IFF artpulseV(ci)))
[-2]  x!1 > TIMEOUT
[-3]  Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-4]  senseV(ci!1, PG)(x!1 + tt!1)
[-5]  senseV(ci!1, PG)(tt!1)
[-6]  RR > TIMEOUT
[-7]  VVI(ci!1, PG)(tt!1)
[-8]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   senseV(ci!1, PG)(TIMEOUT + tt!1)
```

%We use the TRIO+ class feature for axioms. We know that axiom in {-1} is true for all instances of type PG. By the way from the system class we know that the system has only one PG instance called *PG.* For this reason we can instantiate the axiom in {-1} with *(ci!1,PG)* where *ci!1* is the instance of the system *pacemaker_class*

```
Rerunning step: (inst -1 "(ci!1,PG)")
Instantiating the top quantifier in -1 with the terms:
 (ci!1,PG),
this simplifies to:
utility1.1.1 :

{-1}  Alw(VVI(ci!1, PG) IMPLIES
        (Lasted(NOT senseV(ci!1, PG), TIMEOUT) IFF artpulseV(ci!1, PG)))
[-2]  x!1 > TIMEOUT
[-3]  Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-4]  senseV(ci!1, PG)(x!1 + tt!1)
[-5]  senseV(ci!1, PG)(tt!1)
[-6]  RR > TIMEOUT
[-7]  VVI(ci!1, PG)(tt!1)
[-8]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   senseV(ci!1, PG)(TIMEOUT + tt!1)
```

%...Omitted passages...%
%From ax_vvi after some passages we know that an artificial pulse will be provided at time tt! 1+TIMEOUT

```
utility1.1.1.1 :

[-1]  artpulseV(ci!1, PG)(TIMEOUT + tt!1)
  |-------
```

[1] senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
%We know thanks to the system class connections that *artpulseV* is equivalent to the *gen* command for the Actuator class. More in detail we are interested in the instance of the Actuator that is responsible of the ventricle chamber. So for our system class instance *ci!1* is true that *artpulseV(ci!1, PG) = gen(ci!1, actuatorV)*
utility1.1.1.1 :

{-1} artpulseV(ci!1, PG) = gen(ci!1, actuatorV)
[-2] gen(ci!1, actuatorV)(tt!1 + TIMEOUT) AND
      UpToNow(NOT ImpOn(ci!1, actuatorV))(tt!1 + TIMEOUT)
      IMPLIES ImpOn(ci!1, actuatorV)(tt!1 + TIMEOUT)
[-3] artpulseV(ci!1, PG)(TIMEOUT + tt!1)
  |-------
[1] senseV(ci!1, PG)(TIMEOUT + tt!1)

%In order to prove that we can generate an artificial pulse, we have to prove that we are not already providing it. This cause another split for the tree of the proof. We will omit these passages but intuitively we can understand that if we are already providing an artificial pulse then soon in the past we had a ventricular contraction. But this is in contradiction with the fact that we are providing now an artificial pulse command, means that is *too early* to provide another artificial pulse.
Rerunning step: (case "NOT UpToNow(NOT ImpOn(ci!1, actuatorV))(tt!1 + TIMEOUT)")
Case splitting on
  NOT UpToNow(NOT ImpOn(ci!1, actuatorV))(tt!1 + TIMEOUT),
this yields 2 subgoals:
utility1.1.1.1.1 :

[-1] artpulseV(ci!1, PG) = gen(ci!1, actuatorV)
[-2] gen(ci!1, actuatorV)(tt!1 + TIMEOUT) AND
      UpToNow(NOT ImpOn(ci!1, actuatorV))(tt!1 + TIMEOUT)
      IMPLIES ImpOn(ci!1, actuatorV)(tt!1 + TIMEOUT)
[-3] artpulseV(ci!1, PG)(TIMEOUT + tt!1)
  |-------
{1} UpToNow(NOT ImpOn(ci!1, actuatorV))(tt!1 + TIMEOUT)
[2] senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
%After we have proved that we are not already providing an artificial pulse we have all the hypothesis to assert that we will provide an artificial pulse at *tt!1+TIMEOUT*
utility1.1.1.1.2 :

{-1} UpToNow(NOT ImpOn(ci!1, actuatorV))(tt!1 + TIMEOUT)
[-2] artpulseV(ci!1, PG) = gen(ci!1, actuatorV)
[-3] gen(ci!1, actuatorV)(tt!1 + TIMEOUT) AND
      UpToNow(NOT ImpOn(ci!1, actuatorV))(tt!1 + TIMEOUT)
      IMPLIES ImpOn(ci!1, actuatorV)(tt!1 + TIMEOUT)
[-4] artpulseV(ci!1, PG)(TIMEOUT + tt!1)
  |-------
[1] senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
%Because ImpOn is true now this will cause electrical signal on the ventricle chamber (*sensesignalV*) to be true, so the PG will acquire this information. In order to label this signal as a ventricular contraction (eg. the first time instant in which we sense an electrical signal in the ventricle) we have to prove that we are not ignoring now the electrical signals coming from the lead.
utility1.1.1.1.2 :

{-1} Alw(sensesignalV(ci!1, PG) AND NOT ignoresignalV(ci!1, PG) IFF
        senseV(ci!1, PG))
[-2] sense(ci!1, elV) = sensesignalV(ci!1, PG)
[-3] sense(ci!1, elV)(TIMEOUT + tt!1)
[-4] ImpOn(ci!1, actuatorV) = ImpOn(ci!1, elV)
[-5] ImpOn(ci!1, actuatorV)(TIMEOUT + tt!1)
  |-------
[1] senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
%But if we are ignoring the signal then soon in the past (*x!2*) we have encountered a ventricular contraction. But this is in contradiction with the fact that now (*tt!1+TIMEOUT*) we provide an artificial pulse command.

utility1.1.1.1.2.1 :

[-1] senseV(ci!1, PG)(TIMEOUT - x!2 + tt!1)
[-2] ignoresignalV(ci!1, PG)(TIMEOUT + tt!1)
[-3] sense(ci!1, elV) = sensesignalV(ci!1, PG)
[-4] sense(ci!1, elV)(TIMEOUT + tt!1)
  |-------
[1]   ignoresignalV(ci!1, PG)(TIMEOUT - x!2 + VRP + tt!1)
[2]   senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
%In order for the contradiction to succeed we need to state that TIMEOUT>VRP. This is one of the interesting not obvious constraints that results from the specification.
utility1.1.1.1.2.1.2 :

{-1} TIMEOUT > VRP + EPS
[-2] VRP = 320
[-3] x!2 >= 0
[-4] x!2 > 0
[-5] x!2 <= VRP
[-6] senseV(ci!1, PG)(TIMEOUT - x!2 + tt!1)
  |-------
[1]   x!2 < TIMEOUT
[2]   ignoresignalV(ci!1, PG)(TIMEOUT - x!2 + VRP + tt!1)
[3]   senseV(ci!1, PG)(TIMEOUT + tt!1)

Rerunning step: (assert)
Simplifying, rewriting, and recording with decision procedures,
This completes the proof of utility1.1.1.1.2.1.2.
This completes the proof of utility1.1.1.1.2.1.

%Finally we have reached the point in which we have proved that there is a ventricular contraction at time *tt!1+TIMEOUT*
utility1.1.1.1.2.2 :

[-1] sensesignalV(ci!1, PG)(tt!1 + TIMEOUT) AND
      NOT ignoresignalV(ci!1, PG)(tt!1 + TIMEOUT)
      IFF senseV(ci!1, PG)(tt!1 + TIMEOUT)
[-2] sense(ci!1, elV) = sensesignalV(ci!1, PG)
[-3] sense(ci!1, elV)(TIMEOUT + tt!1)
  |-------
{1}   ignoresignalV(ci!1, PG)(tt!1 + TIMEOUT)
[2]   senseV(ci!1, PG)(TIMEOUT + tt!1)

Rerunning step: (assert)
Simplifying, rewriting, and recording with decision procedures,
This completes the proof of utility1.1.1.1.2.2.
This completes the proof of utility1.1.1.1.2.
This completes the proof of utility1.1.1.1.

%Now that we proved that we have a ventricular contraction a *tt!1+TIMEOUT* this is in contradiction with {-1}, so it is obvious that x!1<=TIMEOUT
utility1.1.1.2 :

[-1] Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-2] VVI(ci!1, PG)(TIMEOUT + tt!1)
  |-------
{1}   artpulseV(ci!1, PG)(TIMEOUT + tt!1)
{2}   Lasted(NOT senseV(ci!1, PG), TIMEOUT)(TIMEOUT + tt!1)
[3]   senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
This completes the proof of utility1.1.1.2.
This completes the proof of utility1.1.1.

%Now we take in consideration the case of x!1<=TIMEOUT
utility1.1.2 :

[-1] Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-2] senseV(ci!1, PG)(x!1 + tt!1)
[-3] senseV(ci!1, PG)(tt!1)

[-4] RR > TIMEOUT
[-5] VVI(ci!1, PG)(tt!1)
[-6] Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
{1}   x!1 > TIMEOUT
[2]   senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
%A contraction at time *tt!1+x!1* can be provided by the heart or by the pacemaker {-1}, but because x!1 is <=TIMEOUT this will be a contradiction in both cases.
utility1.1.2 :

{-1} natpulseV(ci!1, Heart)(tt!1 + x!1) OR artpulseV(ci!1, PG)(tt!1 + x!1)
[-2] Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
{-3} senseV(ci!1, PG)(tt!1 + x!1)
[-4] senseV(ci!1, PG)(tt!1)
[-5] RR > TIMEOUT
[-6] VVI(ci!1, PG)(tt!1)
[-7] Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   x!1 > TIMEOUT
[2]   senseV(ci!1, PG)(TIMEOUT + tt!1)

%First we consider the case of a natural pulse at time *tt!1+x!1*
Rerunning step: (case "natpulseV(ci!1,Heart)(tt!1+x!1)")
Case splitting on
   natpulseV(ci!1, Heart)(tt!1 + x!1),
this yields 2 subgoals:
utility1.1.2.1 :

{-1} natpulseV(ci!1, Heart)(tt!1 + x!1)
[-2] natpulseV(ci!1, Heart)(tt!1 + x!1) OR artpulseV(ci!1, PG)(tt!1 + x!1)
[-3] Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-4] senseV(ci!1, PG)(tt!1 + x!1)
[-5] senseV(ci!1, PG)(tt!1)
[-6] RR > TIMEOUT
[-7] VVI(ci!1, PG)(tt!1)
[-8] Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   x!1 > TIMEOUT
[2]   senseV(ci!1, PG)(TIMEOUT + tt!1)

%...Omitted passages...%
%This is the last step of this branch of the tree that cause the contradiction, in fact x*!1* must be <=*TIMEOUT,* but because of a natural pulse at *x!1+tt!1* we also have that *x!1>=RR,* but this is in contradiction with the fact that RR>TIMEOUT
utility1.1.2.1.2 :

{-1} RR = HAV + HPV
[-2] x!1 >= 0
[-3] x!1 > 0
[-4] natpulseV(ci!1, Heart)(tt!1 + x!1)
[-5] TRUE OR artpulseV(ci!1, PG)(tt!1 + x!1)
[-6] Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-7] senseV(ci!1, PG)(tt!1 + x!1)
[-8] senseV(ci!1, PG)(tt!1)
[-9] RR > TIMEOUT
[-10] VVI(ci!1, PG)(tt!1)
[-11] Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   x!1 < HAV + HPV
[2]   x!1 > TIMEOUT
[3]   senseV(ci!1, PG)(TIMEOUT + tt!1)

Rerunning step: (assert)
Simplifying, rewriting, and recording with decision procedures,
his completes the proof of utility1.1.2.1.2.
This completes the proof of utility1.1.2.1.

%...Omitted passages...%
%Now we consider the case in which the ventricular contraction at *tt!1+x!1* has been provided by an

artificial pulse of the PG. In this case the contradiction come from the fact that we provide an artificial pulse at *tt!1+x!1* but it is not possible when *x!1<TIMEOUT* because we already have a ventricular contraction at *tt!1.*
utility1.1.2.2 :

```
{-1}  artpulseV(ci!1, PG)(tt!1 + x!1)
[-2]  Lasts(NOT senseV(ci!1, PG), x!1)(tt!1)
[-3]  senseV(ci!1, PG)(tt!1 + x!1)
[-4]  senseV(ci!1, PG)(tt!1)
[-5]  RR > TIMEOUT
[-6]  VVI(ci!1, PG)(tt!1)
[-7]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
[1]   natpulseV(ci!1, Heart)(tt!1 + x!1)
[2]   x!1 > TIMEOUT
[3]   senseV(ci!1, PG)(TIMEOUT + tt!1)
```

%...Omitted passages...%
This completes the proof of utility1.1.2.2.
This completes the proof of utility1.1.2.
This completes the proof of utility1.1.

%Finally we are back to the main split of the proof. We consider now the right branch where we need to prove that *Lasts(NOT senseV(ci!1, PG), TIMEOUT)(tt!1)*
utility1.2 :

```
[-1]  senseV(ci!1, PG)(tt!1)
[-2]  RR > TIMEOUT
[-3]  VVI(ci!1, PG)(tt!1)
[-4]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
{1}   Lasts(NOT senseV(ci!1, PG), TIMEOUT)(tt!1)
```

%...Omitted passages...%
% In order to prove the consequence, suppose by absurd that we have a ventricular contraction before TIMEOUT, lets call this time instant *tt!1+it!1.* Then this contraction can be provided by a natural pulse, or an artificial pulse. In both cases this will throw a contradiction.
utility1.2 :

```
{-1}  natpulseV(ci!1, Heart)(it!1 + tt!1) OR
      artpulseV(ci!1, PG)(it!1 + tt!1)
[-2]  0 < it!1
[-3]  it!1 < TIMEOUT
[-4]  senseV(ci!1, PG)(it!1 + tt!1)
[-5]  senseV(ci!1, PG)(tt!1)
[-6]  RR > TIMEOUT
[-7]  VVI(ci!1, PG)(tt!1)
[-8]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
```

% We consider the case of natural pulse at *tt!1+it!1*
Rerunning step: (case "natpulseV(ci!1,Heart)(it!1+tt!1)")
Case splitting on
  natpulseV(ci!1, Heart)(it!1 + tt!1),
this yields 2 subgoals:
utility1.2.1 :

```
{-1}  natpulseV(ci!1, Heart)(it!1 + tt!1)
[-2]  natpulseV(ci!1, Heart)(it!1 + tt!1) OR
      artpulseV(ci!1, PG)(it!1 + tt!1)
[-3]  0 < it!1
[-4]  it!1 < TIMEOUT
[-5]  senseV(ci!1, PG)(it!1 + tt!1)
[-6]  senseV(ci!1, PG)(tt!1)
[-7]  RR > TIMEOUT
[-8]  VVI(ci!1, PG)(tt!1)
[-9]  Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
  |-------
```

%...Omitted passages...%
% The heart according to his model provides a natural pulse if and only if it do not sense any ventricular

contraction for a time interval of RR=1/HeartNaturalRate. But this is in contradiction with the fact that we have a ventricular contraction at *tt!1* and RR>TIMEOUT and *it!1<TIMEOUT.* In other words the heart its providing a pulse too early.
utility1.2.1 :

{-1} Lasted_ie(NOT senseV(ci!1, Heart), HAV + HPV)(it!1 + tt!1)
[-2] natpulseV(ci!1, Heart)(it!1 + tt!1)
[-3] TRUE OR artpulseV(ci!1, PG)(it!1 + tt!1)
[-4] 0 < it!1
[-5] it!1 < TIMEOUT
[-6] senseV(ci!1, PG)(it!1 + tt!1)
[-7] senseV(ci!1, PG)(tt!1)
[-8] RR > TIMEOUT
[-9] VVI(ci!1, PG)(tt!1)
[-10] Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
|-------

%...Omitted passages...%
% Here is the step in which the contradiction applies. See [-1], [-5], [-8], [1]
utility1.2.1.2 :

{-1} RR = HAV + HPV
[-2] natpulseV(ci!1, Heart)(it!1 + tt!1)
[-3] TRUE OR artpulseV(ci!1, PG)(it!1 + tt!1)
[-4] 0 < it!1
[-5] it!1 < TIMEOUT
[-6] senseV(ci!1, PG)(it!1 + tt!1)
[-7] senseV(ci!1, PG)(tt!1)
[-8] RR > TIMEOUT
[-9] VVI(ci!1, PG)(tt!1)
[-10] Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
|-------
[1] it!1 < HAV + HPV

Rerunning step: (assert)
Simplifying, rewriting, and recording with decision procedures,
This completes the proof of utility1.2.1.2.
This completes the proof of utility1.2.1.

%...Omitted passages...%
% Now we are interested in the case of artificial pulse at time *it!1+tt!1*
utility1.2.2 :

{-1} artpulseV(ci!1, PG)(it!1 + tt!1)
[-2] 0 < it!1
[-3] it!1 < TIMEOUT
[-4] senseV(ci!1, PG)(it!1 + tt!1)
[-5] senseV(ci!1, PG)(tt!1)
[-6] RR > TIMEOUT
[-7] VVI(ci!1, PG)(tt!1)
[-8] Lasts_ii(VVI(ci!1, PG), TIMEOUT)(tt!1)
|-------
[1] natpulseV(ci!1, Heart)(it!1 + tt!1)

%...Omitted passages...%
% According to the VVI functioning mode the PG provides an artificial pulse if and only if it do not sense any ventricular contraction for a time interval of TIMEOUT. But *it!1<TIMEOUT* so it is not possible to have an artificial pulse at *it!1+tt!1* because we already sense a ventricular contraction a *tt!1.*
utility1.2.2 :

{-1} Lasted(NOT senseV(ci!1, PG), TIMEOUT)(it!1 + tt!1)
[-2] artpulseV(ci!1, PG)(it!1 + tt!1)
[-3] 0 < it!1
[-4] it!1 < TIMEOUT
[-5] senseV(ci!1, PG)(it!1 + tt!1)
[-6] senseV(ci!1, PG)(tt!1)
[-7] RR > TIMEOUT
[-8] VVI(ci!1, PG)(tt!1)
[-9] VVI(ci!1, PG)(it!1 + tt!1)
|-------
[1] natpulseV(ci!1, Heart)(it!1 + tt!1)

% We open the TRIO operator Lasted and we obtain {1}. Now {-5} and {1} completes the proof.
utility1.2.2 :

```
[-1]  artpulseV(ci!1, PG)(it!1 + tt!1)
[-2]  0 < it!1
[-3]  it!1 < TIMEOUT
[-4]  senseV(ci!1, PG)(it!1 + tt!1)
[-5]  senseV(ci!1, PG)(tt!1)
[-6]  RR > TIMEOUT
[-7]  VVI(ci!1, PG)(tt!1)
[-8]  VVI(ci!1, PG)(it!1 + tt!1)
  |-------
{1}   senseV(ci!1, PG)(-it!1 + it!1 + tt!1)
[2]   natpulseV(ci!1, Heart)(it!1 + tt!1)

Rerunning step: (assert)
Simplifying, rewriting, and recording with decision procedures,
This completes the proof of utility1.2.2.
This completes the proof of utility1.2.
```

% The proof is completed
Q.E.D.

4.3 Zot model checker

Zot [7] is an agile and easily extensible bounded model checker. The tool supports different logic languages through a multi-layered approach: its core uses PLTL, and on top of it a decidable predicative fragment of TRIO is defined. An interesting feature of Zot is its ability to support different encodings of temporal logic as SAT problems by means of plugins. This approach encourages experimentation, as plug-ins are expected to be quite simple, compact (usually around 500 lines of code), easily modifiable, and extensible.

Zot offers three basic usage modalities:

- Bounded satisfiability checking (BSC): given as input a specification formula, the tool returns a (possibly empty) history (i.e., an execution trace of the specified system) which satisfies the specification. An empty history means that it is impossible to satisfy the specification.
- Bounded model checking (BMC): given as input an operational model of the system, the tool returns a (possibly empty) history (i.e., an execution trace of the specified system) which satisfies it.
- History checking and completion (HCC): The input file can also contain a partial (or complete) history H. In this case, if H complies with the specification, then a completed version of H is returned as output, otherwise the output is empty.

4.3.1 Satisfiability

This axiomatization written in Zot represents the final result of a long iterative process of adding more and more details taken from the requirements (e.g. the definition of the refractory period and the modularization of the system). The system written as follows is satisfiable for a bounded set of 40 time discrete instants. An example of a possible model produced as output by ZOT is the following:

```
------ time 0 ------
 DDD

------ time 1 ------
 DDD

------ time 2 ------
 DDD

------ time 3 ------
 DDD

------ time 4 ------
 DDD

------ time 5 ------
 DDD
 NATPULSEA

------ time 6 ------
 DDD
 SENSEA
 SENSESIGNALA
 PWAVE

------ time 7 ------
 DDD
 IGNORESIGNALA_ARP
 SENSESIGNALA
 PWAVE

------ time 8 ------
 ARTPULSEV
 DDD
 IGNORESIGNALA_ARP

------ time 9 ------
 DDD
 SENSEV
 SENSESIGNALV
 IMPONV

------ time 10 ------
 DDD
 IGNORESIGNALV
 IGNORESIGNALA_PVARP
 SENSESIGNALV
 IMPONV

------ time 11 ------
 DDD
 IGNORESIGNALV
 IGNORESIGNALA_PVARP

------ time 12 ------
 DDD

------ time 13 ------
 DDD

------ time 14 ------
 DDD

------ time 15 ------
 DDD
 NATPULSEA

------ time 16 ------
 DDD
 SENSEA
 SENSESIGNALA
 PWAVE

------ time 17 ------
 DDD
 IGNORESIGNALA_ARP
 SENSESIGNALA
 PWAVE

------ time 18 ------
 ARTPULSEV
 DDD
 IGNORESIGNALA_ARP

------ time 19 ------
 DDD
 SENSEV
 SENSESIGNALV
 IMPONV

------ time 20 ------
 DDD
 IGNORESIGNALV
 IGNORESIGNALA_PVARP
 SENSESIGNALV
 IMPONV

------ time 21 ------
 DDD
 IGNORESIGNALV
 IGNORESIGNALA_PVARP

------ time 22 ------
 DDD

------ time 23 ------
 DDD
```

In this case at instant 5 a natural pulse of the atrium succeeds, in the next time instant the atrial pulse is sensed, and after 2 time instants (that is the AV delay) an artificial pulse on the ventricle is performed.
In a critical system, as the pacemaker is, the bounded satisfiability is not a sufficient result, because some important properties needs to be valid for all possible state spaces and even in continuous time. For this reason these properties needs to be logically proved, and for this reason another tool is needed: PVS.

5. Pacemaker simulation

For the purpose of this project a JAVA simulation of the pacemaker system has been implemented. Given as input the desired parameters of the patient heart, and the set of the pacemaker parameters, the program produce in output an ECG showing the corresponding behavior of the pacemaker on the patient heart. It is possible to set the parameters of the simulated heart in order to verify all the possible situations that can occurs. Of course, in a real system, the DCM will not have the heart parameters, that will be measured by the physician.
The goal of the simulation is to have a concrete feedback on the system axiomatization. Moreover the JAVA code takes in consideration the real architecture of the system, in order to make a possible reuse of this code when the real hardware will be implemented. The code directly follows from the formal specification. In the following picture you can see the same classes of the PVS specification. Electrical leads are not present because they are not needed for the scope of simulation. The device control monitor for the physician has been included, in order to set the mode and the objective rate of the pacemaker.

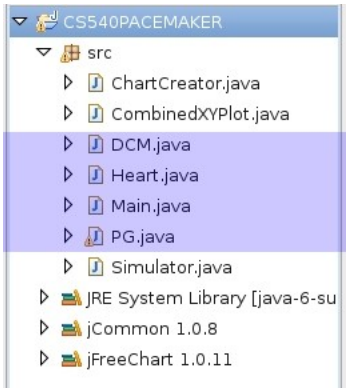


Figure 6: System classes

The DCM (Device Control Monitor) is represented with a GUI where it is possible to set the following parameters:

1 Heart Parameters

HAV: Heart natural distance between an atrial contraction and a ventricular contraction
RR: Distance between two ventricular contraction
PWAVEDUR: PWave duration in milliseconds
PWAVEAMP: Pwave amplitude in millivolts
QRSWAVEDUR: QRSWave duration in milliseconds
QRSWAVEAMP: QRSWave amplitude in milliseconds
ARTWAVEDUR_A: Artificial wave duration that follows from atrium stimulation
ARTWAVEAMP_A: Artificial wave amplitude that follows from atrium stimulation
ARTWAVEDUR_V: Artificial wave duration that follows from ventricular stimulation
ARTWAVEAMO_V: Artificial wave amplitude that follows from ventricular stimulation

2 Pacemaker Parameters

Functioning Mode: VVI, AAT, DDD
LRL Timeout: Timeout for artificial stimulation according
to the functioning mode
VRP: Refractory period after ventricular stimulation
ARP: Refractory period after atrial stimulation
PVARP: Refractory period after ventricular stimulation listening for atrial events

3 Common VVI Situations

No Disease heart: Settings for a safe heart
Atrial Fibrillation: Settings for heart suffering of atrial fibrillation

The output of the simulation will be an ECG that shows how the heart and the pacemaker combine their behavior. The advantage of this simulation environment is that it adds another level of realism to the problem, which is the Electrocardiogram signal that has been modeled in the PVS specification with a binary value (eg: present, not present), while here it assumes a range of values.

6 Testing

The Java simulator is implemented in order to easily create significant test cases that verifies how a real implementation meets the formal requirements.
The test cases presented are related to the system properties proved thanks to the PVS tool, and are the following:

- Test Case 1: Security Property, under normal behavior of the patient heart.
- Test Case 2: Utility Property, when both chambers are artificially paced.
- Test Case 3: Utility Property, when only the ventricle is artificially paced.

For verifying the test cases is used what it would be the output of an electrocardiogram as the output of the simulator. This version of the electrocardiogram represents heart P wave and QRS wave as triangular signal and the pacemaker electrical discharge as a spike, a negative impulse.

6.1 Test Case 1: Security

PVS definition:
CONJECTURE Alw(senseV AND RR<Timeout AND DDD AND Lasts_ii(DDD, RR) IMPLIES Futr(senseV, RR) AND Lasts(NOT senseV, RR))

Parameters set:
HAV=200ms; HPV=700 ms; RR= HAV+HPV=900ms; Timeout=1100 ms;

Expected Result:
The pacemaker must not interfere with the natural heart rate. In the electrocardiogram no spike should be present.

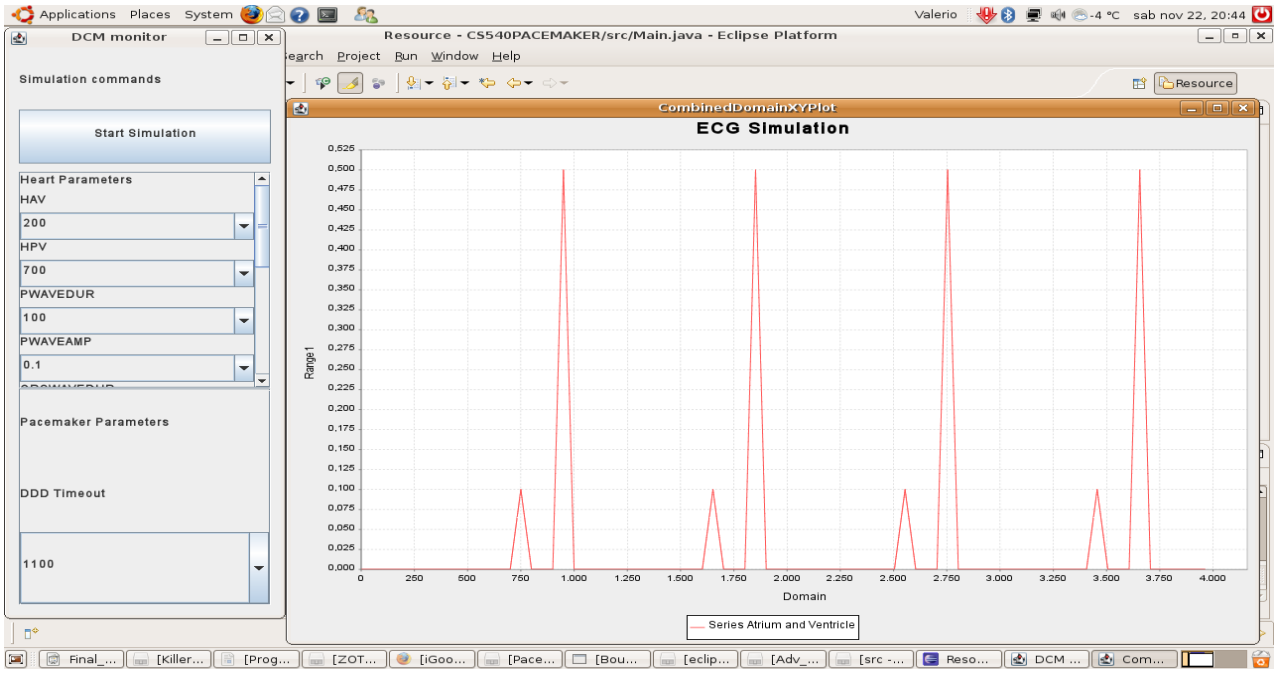


Figure 7: Test case 1 in the simulator

Test Case 1 confirms the expected result.

6.2 Test Case 2: Utility

PVS definition:
CONJECTURE Alw(senseV AND RR>Timeout AND DDD AND Lasts_ii(DDD, Timeout) IMPLIES Futr(senseV, Timeout) AND Lasts(NOT senseV, Timeout))

Parameters set:
HAV=300ms; HPV=1200 ms; RR= HAV+HPV=1500ms; Timeout=800 ms; AV_delay=200ms

Expected Result:

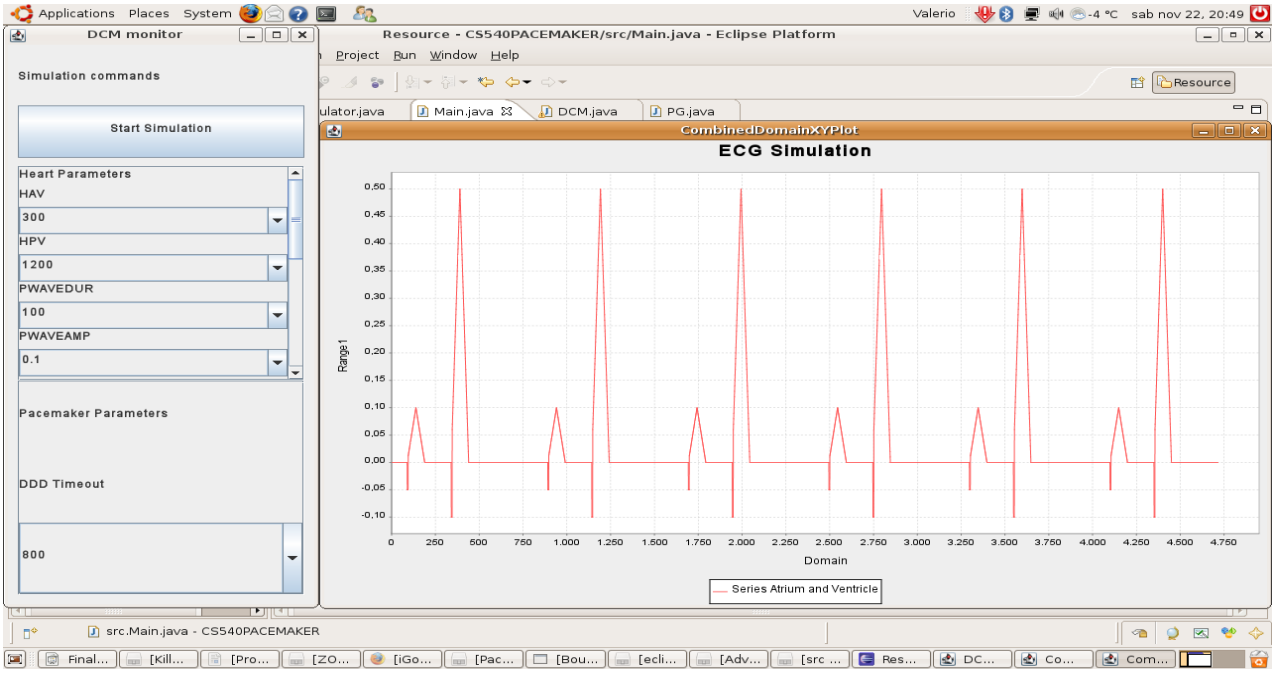


Figure 8: Test case 2 in the simulator

The pacemaker should pace both chambers. The atrium because RR>Timeout and ventricle because HAV>AV_delay. Pacemaker spikes both in the atrial and the ventricular signal.Test Case 2 confirms the expected result.

6.3 Test Case 3: Utility

PVS definition:
CONJECTURE Alw(senseV AND RR>Timeout AND DDD AND Lasts_ii(DDD, Timeout) IMPLIES Futr(senseV, Timeout) AND Lasts(NOT senseV, Timeout))

Parameters set:
HAV=300ms; HPV=700 ms; RR= HAV+HPV=1000ms; Timeout=1100 ms; AV_delay=200ms

Expected Result:
The pacemaker should pace the ventricle because HAV>AV_delay but not the atrium because RR<Timeout. Pacemaker spikes only in the ventricular signal.

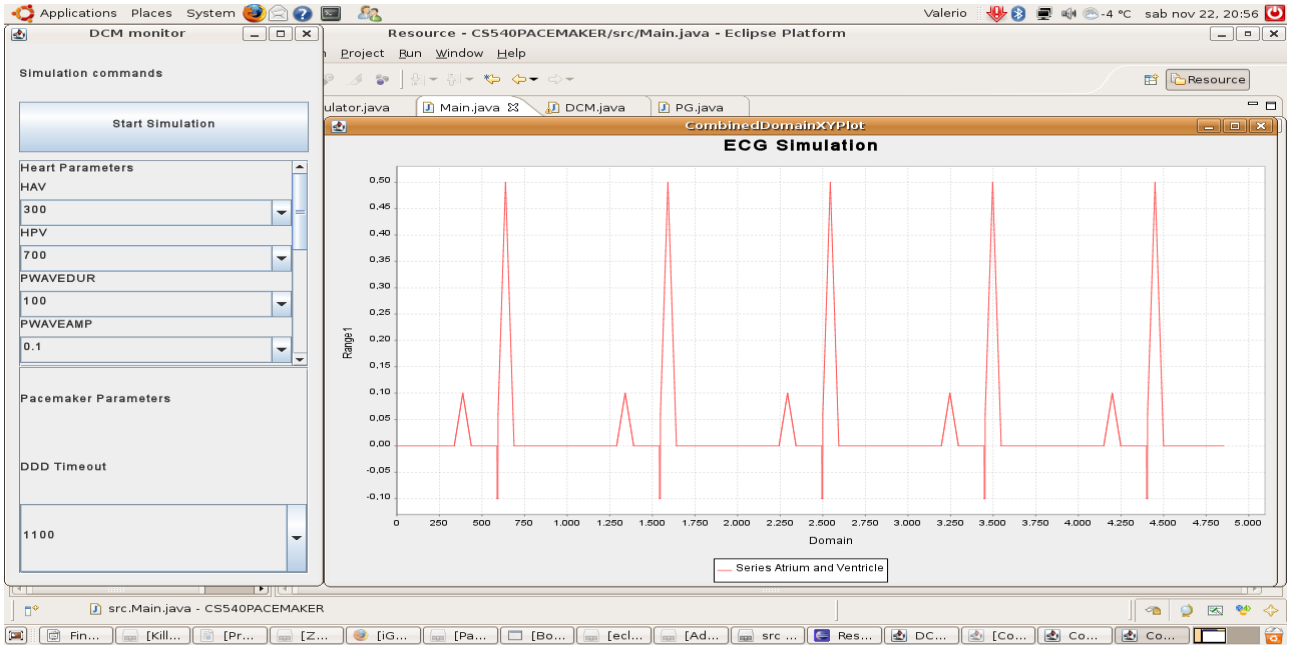


Figure 9: Test case 3 in the simulator

Test Case 3 confirms the expected result.

7 Project Management

7.1. Organization

The team members are:

* Valerio Panzica La Manna.
* Andrea Tommaso Bonanno
* Alfredo Motta

All team members participated and gave their contribute throughout all the steps of the project realized. In particular, once the initial system requirements were fixed formally each team member cared the work about one specific mode of the pacemaker to be implemented.
The related work was divided in the following way:

* Valerio worked on DDD.
* Andrea worked on AAT;
* Alfredo worked on VVI;

Finally all the different parts developed in PVS and ZOT were merged and the new integrated system behavior was tested again.
For the work concerning the Java Simulator all the team members participated to the design and coding phases, each taking care of the implementation of the specific mode assigned to him.

7.2. Project Timeline

The timeline followed by the project was the following:

* **09/30/2008** Requirements : The initial requirements have been fixed as a TRIO specification. The system requirements have been studied and formalized during the second half of September.
* **11/15/2008** Formal requirements have been verified by means of Zot and PVS. The month of October and the first half of November have been spent by each team member to iteratively work on and refine the system requirements as well as to coordinate the work in order to synchronize the design of the final whole system.
* **11/30/2008** A Test Plan has been produced in order to figure out what to focus on for the Simulator implementation.
* **12/10/2008** Design and Coding: Implementation of the Java Simulator.
* **12/20/2008** Testing on the simulator.
* **01/14/2009** Summary report.

8 Conclusions

The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to the reliability and robustness of the system. The formal description of the system can be used to guide further development activities, additionally, it can be used to verify that the requirements for the system being developed have been completely and accurately specified. The field of formal methods has its critics. At the current state of the art, proofs of correctness, whether handwritten or computer-assisted, need significant time (and thus money) to produce, with limited utility other than assuring correctness. This makes formal methods more likely to be used in fields where the benefits of having such proofs, or the danger in having undetected errors, makes them worth the resources. The pacemaker industry is one of the best example of a life threatening field together with aerospace engineering.
Our main aim regarding this project was to understand how to exploit TRIO's expressive power in a big real project and what are the pros and cons of using this kind of approach. The big advantage of the language and of the tools used is that they help us to develop a more complete understanding of the system we are designing. When an error in the specification occurs, it is easy to understand what is wrong. The final result, even if very difficult to obtain, is that every axiom written and the relations among them are logically correct. This is very rewarding. On the other hand, the process is very time consuming, mainly because of the PVS proof. The ZOT tool is a good compromise between the results obtained(the satisfiability), and the speed to obtain it, thanks to its automatic reasoning. Another improvement to speed up the entire process could be to create a library of the most common self-contained TRIO+ components, already verified in for PVS. In that case it would be possible to reduce the PVS proof only to the relation between these.
Future works for this project includes the coding of the real pacemaker hardware created by University of Minnesota based on the PIC18F4520 developed by MICROCHIP.

References

[1] PACEMAKER Challenge: http://sqrl.mcmaster.ca/pacemaker.htm

[2] SCORE website: http://score.elet.polimi.it/.

[3] Boston Scientific, "PACEMAKER System Specification" , January 3, 2007

[4] Morzenti A., San Pietro P., Object Oriented Logical Specification of Time-Critical System, ACM Transactions on Software Engineering and Metodology, Vol 3, N 1, January 1994

[5] Mandrioli D., "The specification of Real-Time Systems: a Logical and Object Oriented Approach", Proceedings TOOLS 8-USA 1992, Santa Barbara, California

[6] Crow J. : A Tutorial Introduction to PVS. WIFT '95. Boca Raton , Florida, April 1995

[7] A User's Guide to Zot Matteo Pradella CNR IEIIT, Milano, Italy , June 2008

[8] D. Santel , C . Trautmann, W . L i u, "Formal Safety Analysis and the Software Engineering Process in the Pacemaker Industry "

[9] H. Weston Moses, James C. Mullin, *A Practical Guide to Cardiac Pacing*

[10] S. Serge Barold, Roland Stroobandt, Alfons F. Sinnaeve, *Cardiac Pacemakers Step by Step*

[11] Sam Owre and N. Shankar ,*Writing PVS Proof Strategies,* STRATA 2003, Rome, Italy, September 2003