From Verified Models to Verified Code for Safe Medical Devices

Zhihao Jiang

A DISSERTATION

in

Computer and Infomation Science

Presented to the Faculties of the University of Pennsylvania
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

2016

Rahul Mangharam, Associate Professor of Electrical and Systems Engineering
Supervisor of Dissertation

Lyle Ungar, Professor of Computer and Information Science
Graduate Group Chairperson

Dissertation Committee:
Rajeev Alur (Chair), Zisman Professor of Computer and Information Science
Insup Lee, Cecilia Fitler Moore Professor of Computer and Information Science
Oleg Sokolsky, Research Associate Professor of Computer and Information Science
Pieter J. Mosterman, Adjunct Professor at the School of Computer Science at McGill University
Richard Gray, Biomedical Engineer at FDA

From Verified Models to Verified Code for Safe Medical Devices

To my parents.

Acknowledgments

I'm most grateful to my dissertation advisor, Prof. Rahul Mangharam, for his guidance and support throughout my dissertation research. It was my great pleasure to work with him.

I am grateful to Dr. Sanjay Dixit from the Hospital of University of Pennsylvania who shared physiological knowledge which is the backbone of my dissertation.

I would like to express my gratitude to Prof. Insup Lee, Prof. Rajeev Alur and Prof. Oleg Sokolsky for their continuous support and encouragement to my research over the years.

I would like to thank Dr. Pieter J. Mosterman who provided me with valuable insight on model-based design.

I am thankful to Dr. Richard Gray for providing suggestions to my research from the regulatory perspective.

I would also like to thank Dr. Houssam Abbas, Dr. Miroslav Pajic and Dr. Madhur Behl for their help during my dissertation research.

ABSTRACT

From Verified Models to Verified Code for Safe Medical Devices

Zhihao Jiang
Rahul Mangharam

Medical devices play an essential role in the care of patients around the world, and can have a life-saving effect. An emerging category of autonomous medical devices like implantable pacemakers and implantable cardioverter defibrillators (ICD) diagnose conditions of the patient and autonomously deliver therapies. Without trained professionals in the loop, the software component of autonomous medical devices is responsible for making critical therapeutic decisions, which pose a new set of challenges to guarantee patient safety. As regulation effort to guarantee patient safety, device manufacturers are required to submit evidence for the safety and efficacy of the medical devices before they can be released to the market. Due to the closed-loop interaction between the device and the patient, the safety and efficacy of autonomous medical devices must ultimately be evaluated within their physiological context. Currently the primary closed-loop validation of medical devices is in form of clinical trials, in which the devices are evaluated on real patients. Clinical trials are expensive and expose the patients to risks associated with untested devices. Clinical trials are also conducted after device development, therefore issues found during clinical trials are expensive to fix. There is urgent need for closed-loop validation of autonomous medical devices before the devices are used in clinical trials.

In this thesis, I used implantable cardiac devices to demonstrate the applications of model-based approaches during and after device development to provide confidence towards the safety and efficacy of the devices.

A heart model structure is developed to mimic the electrical behaviors of the heart in various heart conditions. The heart models created with the model structure are capable of interacting with implantable cardiac devices in closed-loop and provide physiological interpretation for a large variety of heart conditions

With the heart models, I demonstrated that closed-loop model checking is capable of identifying known and unknown safety violations within the pacemaker design. More importantly, I developed a framework to choose the most appropriate heart models to cover physiological conditions that the pacemaker may encounter, and provide physiological context to counter-examples returned by the model checker.

A model translation tool UPP2SF is then developed to translate the pacemaker design in UPPAAL to Stateflow, and automatically generated to C code. The automated and rigorous translation ensures that the properties verified during model checking still hold in the implementation, which justifies the model checking effort.

Finally, the devices are evaluated with a virtual patient cohort consists of a large number of heart models before evaluated in clinical trials. These in-silico pre-clinical trials provide useful insights which can be used to increase the success rate of a clinical

trial.

The work in this dissertation demonstrated the importance and challenges to represent physiological behaviors during closed-loop validation of autonomous medical devices, and demonstrated the capability of model-based approaches to provide safety and efficacy evidence during and after device development.

Contents

List of Tables

List of Figures

Chapter 1

Medical Devices: Current State and Challenges

The medical device market is worth $289 billion, of which $110 billion is from the US alone, with this number projected to reach $133 billion in 2016 (Research and Markets [2015]). Examples include everything from adhesive bandages, stents, artificial joints, drug infusion pumps to surgical robots, implantable cardiac pacemakers, and devices still undergoing basic research like the artificial pancreas. To take one example of the societal impact of medical devices, an estimated 3 million people worldwide have implanted cardiac pacemakers (a heart rate management device), with 600,000 added annually. Clinical trials have presented evidence that patients implanted with cardiac defibrillators (another heart rate management device) have a mortality rate reduced by up to 31% (Moss et al. [2012]).

The US Food and Drug Administration (FDA) defines a medical device as an instrument, apparatus, implement, machine, or implant which is:

- intended for use in the diagnosis of disease or other conditions, or in the cure, mitigation, treatment, or prevention of disease, in humans or other animals, or
- intended to affect the structure or any function of the human body or other animals, and which does not achieve any of its primary intended purposes through chemical action and which is not dependent upon being metabolized for the achievement of any of its primary intended purposes."

In general, medical devices are categorized according to their risk factors - Class I, Class II and Class III, corresponding to low-risk, medium-risk and high-risk devices (U.S.FDA [2014]). Fig. 1.1 gives an intuitive description of medical devices examples across a range of diagnostic and therapeutic risk.

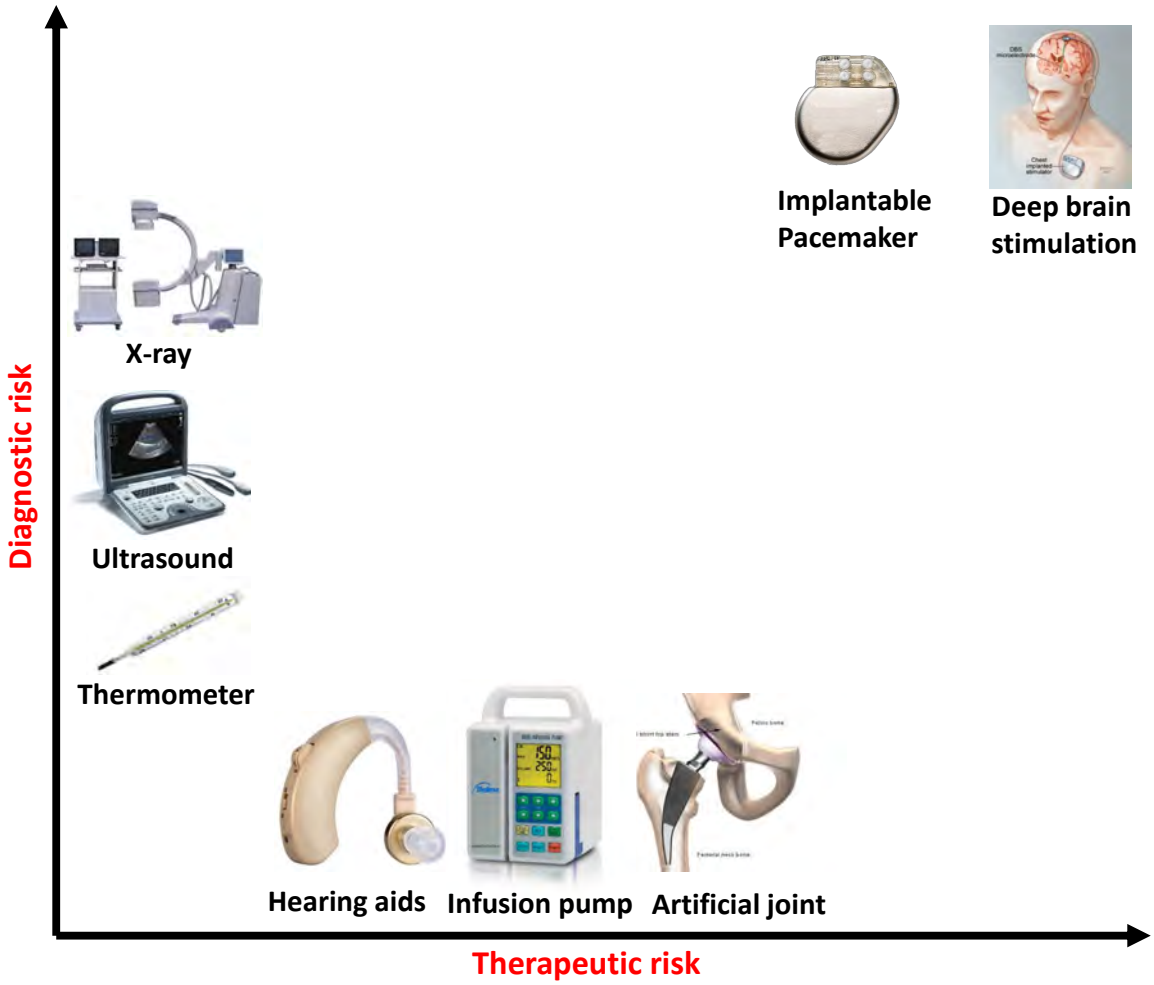


Figure 1.1: Current medical devices across a range of diagnostic and therapeutic risk. Implantable software-controlled devices such as the pacemaker and defibrillator which operate in a closed-loop of sensing, control and actuation are amongst the highest risk

1.1 Closing the Device-Patient Loop

Medical devices operate across a range of invasiveness and intervention with the patient in the loop. For diagnostic-only devices, like an X-ray machine, the medical professional operates the device to obtain patient data. Upon interpretation of the data, the medical professional performs diagnosis followed by delivery of proper therapy to the patient (Fig. 1.2.(a)). For therapy-only devices, e.g. a drug infusion pump, the medical professional configures the device infrequently based on prior diagnosis of the patient so the device only executes the therapy on the patient (Fig. 1.2.(b)). We denote these devices as **Open-loop Medical Devices** as there is no direct feedback between the patient and the device. For open-loop devices, the device operates under the supervision of professionally-trained physicians. The device's safety is mostly determined by how accurately it provides information to the physicians or how faithfully

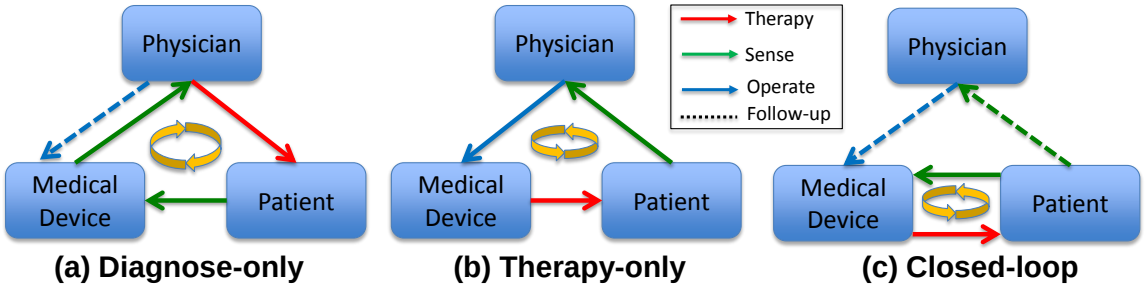


Figure 1.2: Diagnostic-only and therapy-only devices do not interact with the patient in direct closed-loop. The physician is responsible for the diagnostic and/or therapeutic decisions. However in closed-loop medical devices, the devices interact with the patient in closed-loop and have to make therapeutic decisions based on their own diagnosis.

it operates as instructed by the physicians.

There is a class of devices with both diagnostic and therapeutic functions, i.e. implantable cardiac devices to treat cardiac arrhythmia, deep brain stimulation devices (Coffey [2009]) to treat Parkinson's disease and artificial pancreas to treat Type-1 diabetes. These devices capture and diagnose the patient's physiological conditions from patient signals, *and* deliver therapy in response (Fig. 1.2.(c)). These devices usually operate autonomously with very little human intervention. We denote them as **Autonomous Medical Devices**. The benefits of autonomous medical devices are timely therapy and more independent life-style. However, autonomy of these medical devices also brings about safety and efficacy concerns. With open-loop medical devices, the diagnosis and therapy decisions are made by medical professionals, who have expert knowledge of human physiology. Therefore they are able to identify adverse health conditions and adjust the therapy accordingly. On the other hand, closed-loop medical devices have to make both the diagnosis and therapy decisions on their own. The domain expertise required to make those decisions has to be programmed into the device. It is not feasible to encode all the knowledge of human physiology into the device. For unanticipated physiological conditions, when the appropriate response has not been programmed into the device, the device may deliver inappropriate therapy which can have an adverse effect on patient's health. Therefore, these devices are usually classified into the highest risk category and undergo the most stringent regulation.

1.1.1 Challenges for Developing Autonomous medical Devices

There are multiple challenges to develop safe and effective autonomous medical devices:

Physiological Complexity

Human physiology is complex and not completely understood. For instance, the functionality of the heart can be interpreted from *multiple perspectives*: from its electrical activity, mechanical contractions of the heart muscles, and dynamics of blood flow. The physiology of the heart can also be analyzed with *multiple scales*: from the molecular level to cellular level all the way to the organ and system level. It is impossible to encode all these contexts into the device, hence inappropriate diagnosis and therapy are observed due to the lack of physiological contexts (Sandler et al. [2010], Hauser and Maron [2005]).

Physiological Variability

Physiological conditions and parameters demonstrate different levels of variability both within the individual at different times, levels of exertion and under the influence of medication and also across individuals. For instance, a segment of the population may have additional electrical conduction pathways within their heart, which makes them vulnerable to certain heart diseases. Consequently, autonomous medical devices should be able to safely operate under a large variety of physiological conditions. This is difficult to guarantee, as the device designer must consider all possible physiological conditions, and their interaction with the device, during the development of the device.

Limited Observability

Autonomous medical devices normally rely on minimally invasive measurement of the physiological parameters in order to allow the patients to live their normal life. For example, implantable pacemakers and defibrillators commonly have only two leads and therefore two points of observation for the whole heart. The limited observability inevitably leads to ambiguities as different physiological conditions can map to the same input sequence to the device, resulting in inaccurate diagnosis and therapy.

Software-related Medical Device Recalls

The diagnostic and therapeutic functions of the autonomous medical devices are mostly controlled by their software components due to their complexity. For instance, implanted cardiac pacemakers and defibrillators have approximately 80,000-100,000 lines of software code which essentially makes all sensing, control and actuation decisions autonomously within the human body, over the 5-7 year device lifetime [1]. Software embedded in a medical device, unlike electrical and mechanical components, does not fail due to corrosion, fatigue or have statistical failures of sub-components. Software failures are uniquely sourced in the design and development of the system.

[1] Paul L. Jones. Senior Systems/Software Engineer, Office of Science and Engineering Laboratories, U. S. FDA. Personal communication, 2010.

	Software change control	Software Design	Software design manufacturing process	Sum	% of all CDRH recalls
2008	13	141	2	**156**	**18.3%**
2009	9	111	1	**121**	**15.4%**
2010	4	73	3	**80**	**8.9%**
2011	11	182	10	**203**	**15.8%**
2012	12	169	5	**186**	**15.5%**
Sum	**49**	**676**	**21**	**746**	**15.1%**

Figure 1.3: Medical device recalls due to software issues have risen from 10% in the 1990s to ~15% in the past decade (U.S.FDA [2012])

According to the US Food and Drug Administration, in 1996, 10% of all medical device recalls were caused by software-related issues (Maisel et al. [2001]). This percentage rose to an average of 15% of recalls from 2008 to 2012 (Fig. 1.3). Malfunctions of closed-loop medical devices usually have severe consequences, which will be categorized as *Class I*, meaning there is a "reasonable probability that use of these products will cause serious adverse health consequences or death." (U.S.FDA [2006], Zhang et al. [2015], Sandler et al. [2010]).

1.2 Medical Device Regulation Efforts and Challenges

In the United States, the FDA is the primary regulatory authority responsible for assuring the *safety, efficacy and security* of patients using medical devices. Based on the rationale that 1) manufacturers know their devices better than the regulator, and 2) the variety of medical devices requires a variety of approaches, it is the device manufacturers' responsibility to demonstrate the safety and efficacy of the medical devices. Manufacturers are required to complete a pre-market submission before the devices can be released to the market. The level of requirements for the submission is determined by the safety classification of the devices.

1.2.1 Guidelines During Device Development

A set of general guidelines are recommended by the FDA (U.S.FDA [1997, 2002, 2005]) which list the activities that need to be performed during the development of the devices. In safety-critical industries such as automotive electronics, avionics and nuclear systems, international standards are **enforced** for software system development, evaluation, manufacturing and post-market changes (Fürst et al. [2009], Feiler et al. [2010]). This awareness is only beginning to enter the medical device industry as

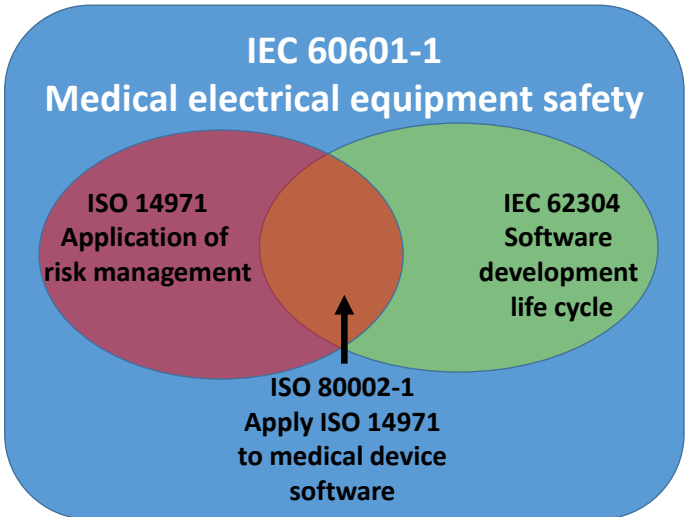


Figure 1.4: International standards for medical device safety. These standards define the required activities during the development process.

compliance with international standards are "recommended" in the aforementioned guidelines (Jetley et al. [2006]).

Fig. 1.4 describes the primary standards to enforce medical device safety and their relationships. The basic rationale behind these standards is that: if all the risks/hazards of the device are identified and reasonably mitigated, and the device is developed with rigorous process, the device is *reasonably safe*. The IEC 60601 Medical Electrical Equipment - General requirements for basic safety and essential performance is a product safety standard that all electronic medical devices must comply to. IEC 60324 specifies the processes and activities needed to perform during the software development life cycle to ensure software safety.

Risk management is a core activity throughout the software development life cycle. ISO 14971 is specified for the application of risk management to medical devices. In addition, for each risk management activity of ISO 14971, ISO 80002-1 provides additional guidelines for the software component, which highlights and explains approaches to assuring that software safety is adequately addressed.

1.2.2 Pre-Market Evaluation with Clinical Trials

Devices that have high risk factors are required to submit clinical evidence for their safety and efficacy, often in form of clinical trials. In clinical trials, the devices are used on a preselected population of patients following carefully-designed protocols. The goal of a medical trial, in part, is to obtain unambiguous results for the primary question of the trial which can support the safety and/or efficacy of the devices. However, conducting clinical trials is very time consuming and expensive, and risks found during clinical trials are very expensive to fix (U.S.FDA [2013]).

1.2.3 Lack of Closed-loop Validation During and After Device Development

While formal methods are used for medical device software validation (Boston Scientific Corporation [2007a], Gomes and Oliveira [2009b], Jee et al. [2010a]), testing continues to be the primary method for safety and efficacy evaluation during device development. Testing for medical device software currently is ad hoc, error prone, and very expensive. Traditional methods of testing do not suffice as the test generation cannot be done independently of the current state of the patient and organ. The primary approach for system-level testing of medical devices is unit testing using a playback of pre-recorded electrogram and electrocardiogram signals (Cortner [2003], Vip [2006]). This tests if the input signal triggers a particular response by the device, but is unable to check how the device affects the physiological conditions of the patient.

The only closed-loop validation activity is clinical trials, which is very expensive and expose the patients to uncertified devices. Issues found during clinical trials are also expensive to fix. There is urgent need for closed-loop validation approaches during and after device development to provide safety and efficacy confidence to the device design.

1.3 Model-based Design and Pre-certification of Medical Devices

In industries like automotive and avionics, mathematical models of the physical environment that the system operates in are developed for analyzing the safety and efficacy of the control systems (or their models) (Fürst et al. [2009], Feiler et al. [2010]). Similar approaches can potentially help during the development process of autonomous medical devices and provide extra confidence to the devices before conducting clinical trials. However, unlike man-made systems like automobiles and aircrafts, physiological systems are less understood with larger variations for the type and degree of patient conditions. The lack of faithful models of physiological environment for the autonomous medical devices is one of the reason that model-based approaches are not well-adopted in the medical device industry.

As computational models of human physiology are developed, they can be used to interact with closed-loop medical devices or their models. The FDA is starting to recognize in-silico modeling and simulation as regulatory-grade evidence for device safety and efficacy. For example, Ghorbani and Bogdan [2013] developed glucose-insulin models that can be used to evaluate control algorithms for artificial pancreas devices which can sense blood glucose and deliver insulin. Simulation results with the models have been recognized by FDA to replace animal trials, in part, which significantly reduced cost (B. P. Kovatchev and M. Breton and C. Dalla Man and C. Cobelli [2009]). With the increasing interest and recognition from the regulators,

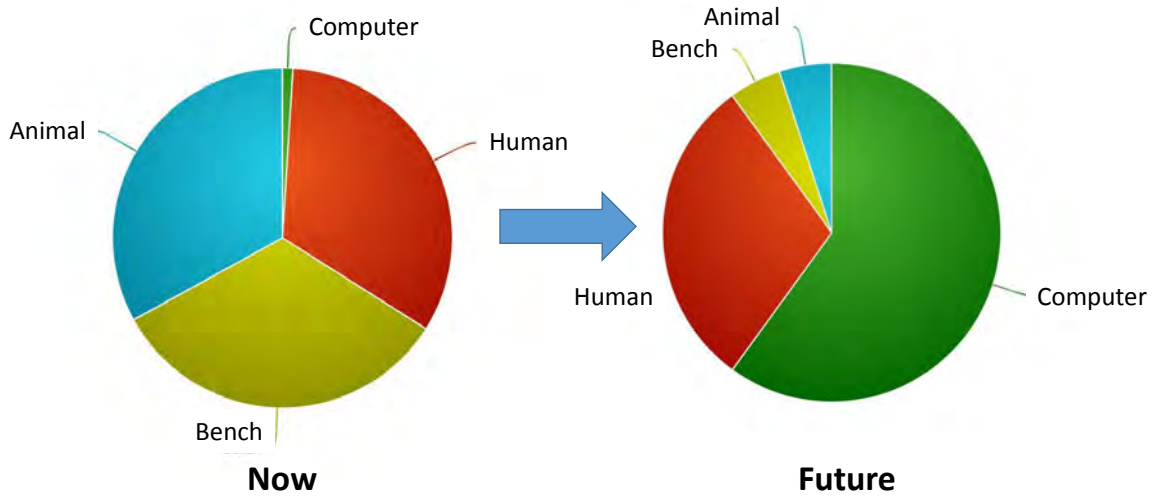


Figure 1.5: Percentage of computer simulation is expected to increase as safety and effectiveness evidence of medical devices (http://mdic.org/)

computer models and simulations are expected to play bigger role as as "regulatory-grade evidence" evidence in the development of future closed-loop medical devices (Fig. 1.5).

1.4 Contributions

In this dissertation, implantable cardiac devices are used as working examples to demonstrate how model-based approaches can help improve the safety and efficacy of autonomous medical devices during and after their development. By demonstrating the process of developing verified models to generate verified code of the devices, the results of model-based closed-loop validation can provide confidence towards the safety and efficacy of the devices.

The contribution of this dissertation is fourfold:

1. **Clinical Electrophysiology heart model structure:** A heart model structure was developed that can represent a large variety of heart conditions and interact with implantable cardiac devices in closed-loop. The heart model structure is available in both software and hardware for different applications of closed-loop validation.

2. **Automated heart model abstraction and refinement:** An abstraction tree structure was developed to abstract the heart models to capture the large variability of heart conditions during closed-loop model checking of implantable pacemaker, and refine the heart models to provide physiological interpretability to the counter-examples.

3. **Automated model translation for code generation:** An automated model translation tool UPP2SF was developed to translate verified UPPAAL device

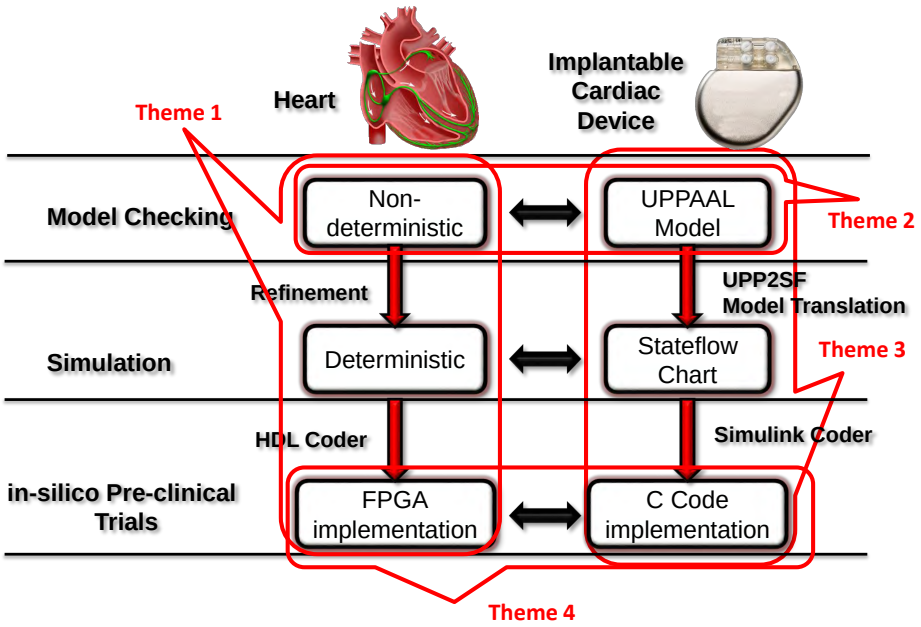


Figure 1.6: Model-based design for implantable cardiac devices with closed-loop validation

model to Stateflow model, which allows rigorous translation from model to C code.

4. **in-silico Pre-clinical trials:** A large virtual population of heart models was generated to evaluate the VT/SVT discrimination algorithms of Implantable Cardioverter Defibrillators before clinical trials.

Fig. 1.6 demonstrates the structure of the dissertation. The dissertation can be broken down into 4 themes, which are illustrated in Fig. 1.6.

The remaining dissertation is arranged as follow: Chapter 2 discusses a dual chamber pacemaker design (Boston Scientific Corporation [2007b]) as the motivating example to illustrate the safety hazards that the device may pose to the patients. Chapter 3 discusses the physiological models that are necessary for closed-loop evaluation of medical devices, and how to use those models to represent complex physiology with large variability Jiang et al. [2012a]. Chapter 4 discusses the use of model-checking techniques to evaluate the safety and efficacy of device design early in the device design stage, with focus on the abstraction and refinement of the heart models to cover large variety of physiological conditions (Jiang et al. [2014]). Chapter 5 discusses the rigorous translation from verified device model to device implementation which maintains the verified properties (Pajic et al. [2012b]). Chapter 6 discusses the in-silico pre-clinical trial, which the devices are evaluated on a virtual patient cohort consists of physiological models to provide useful insights for planning a clinical trial.

1.5 Useful terminologies for often misinterpreted terms

Ensuring the safety and efficacy of complex medical devices has drawn interest not only from stakeholders like regulators and industries, but also medical professionals and academia. Different communities have different interpretations over certain terminologies, often causing misunderstandings. In this dissertation, terminologies from the regulation perspective are adopted. Most of the definitions are referred from the FDA guideline document General Principles of Software Validation (U.S.FDA [2002]). Below are several terminologies that are used throughout the dissertation which worth clarifying.

1.5.1 Requirements vs. Specifications

By the definition of the FDA (U.S.FDA [2005]), the requirements of a system describe **what** the system should achieve and the specifications of a system describe **how** the system is designed to satisfy the requirements. For instance, a requirement for an autonomous car is "The car should not hit objects". The corresponding specification can be "brake if the speed of the car is greater than x and the distance to the object is less than y". It is obvious that a car satisfying its specification may not satisfy the requirement (e.g. when the car is driving too fast or the obstacle pops up right in front of the car). In this dissertation, the word requirement in particular denotes the intended uses of the medical devices to improve physiological conditions.

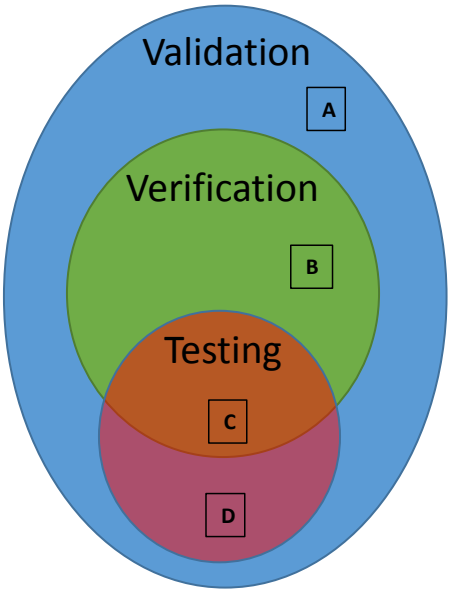
1.5.2 Validation vs. Verification vs. Testing

As defined in U.S.FDA [2002], software validation is the confirmation by examination and provision of objective evidence that:

1. software specifications conform to user needs and intended uses, and
2. the particular requirements implemented through software can be consistently fulfilled

The first aspect ensures the device is safe and effective. The second aspect maintains the traceability of requirements throughout the development life cycle. Software verification fulfills the second aspect of software validation by "providing objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase. "

Testing is the technique that can be used for validation and/or verification. Fig. 1.7 illustrates the relationship between validation, verification and testing, and different activities during the software development life cycle to ensure the safety and effectiveness of the software.



Validation activities	Zone
Planning	A
Requirements	A
Traceability	A,B
Change management	A
User site testing	D
Defect resolution	A,B
Risk management	A
Intended use	A
Evaluations	B
Design reviews	B
System testing	C
Regression testing	C,D

Figure 1.7: Validation activities during the software development life cycle (D A. Vogel [2011])

1.5.3 Closed-loop vs. Open-loop Evaluation

In open-loop evaluation, i.e. open-loop testing, input sequences are send to the system and system outputs are compared with expected outputs. In open-loop testing, the system outputs do not affect the inputs afterward. In closed-loop evaluation, the environment of the system is taken into account. System outputs affect the state of the environment and thus affect the input sequences. For closed-loop medical devices, clinical trials are currently the most common closed-loop evaluation method. Enabling closed-loop evaluation during device design requires models of the environment, which is human physiology for closed-loop medical devices.

Closed-loop evaluation accomplishes two goals in model-based design: 1) It enforces environmental constraints so that the test space is smaller and the test cases have physiological relevance, 2) Execution traces can be better interpreted as the physiological models encode domain knowledge.

Chapter 2

A Motivating Example: A Dual Chamber Pacemaker Design

Intuitively, an autonomous medical device like pacemaker should satisfy the following requirements: 1) improve unhealthy physiological conditions of a patient (efficacy), and 2) not to deteriorate current physiological condition (safety). Evaluating both requirements requires understanding of the physiological environment the device is in. In the following sections, the physiological basis for the heart and the pacemaker is introduced, followed by a dual chamber pacemaker specification (Boston Scientific Corporation [2007b]). Potential safety hazards for a pacemaker were then analyzed and two known safety hazards were discussed in detail. The dual chamber pacemaker specification is then used throughout the thesis to demonstrate how different model-based techniques can be used to validate the safety and efficacy of an autonomous medical device.

2.1 Physiology Basis of the Heart and the Pacemaker

First, we use this small section to introduce the physiology basis of the heart and the application of implantable cardiac devices. Readers with knowledge of this subject can skip to the following sections.

2.1.1 Blood Circulation System

The heart is the "motor" for blood circulation within our body. The heart has two ventricles which pump the blood out of the heart, and two atria which gather blood from the body and pump them into the ventricles. (Fig. 2.1.(a)) There are two circulations through the heart: the *Pulmonary circulation* and the *Stemic circulation*. In the pulmonary circulation, the right atrium collects oxygen-depleted blood from all over the body and pumps it into the right ventricle. The right ventricle then

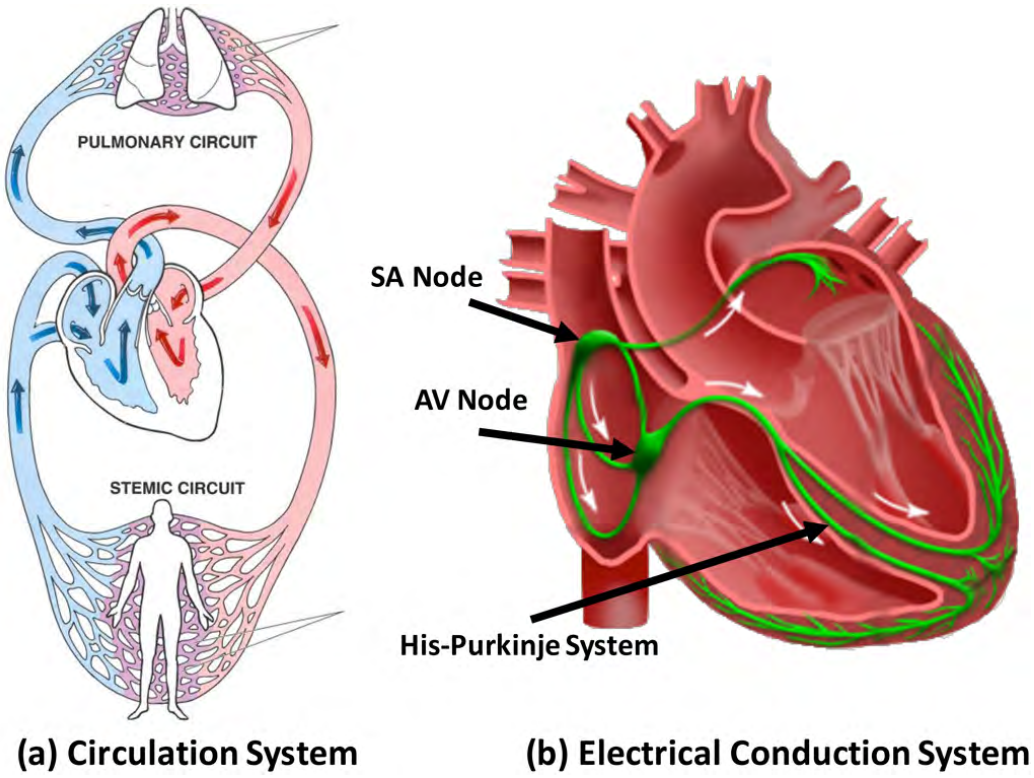


Figure 2.1: (a) The circulation system (http://revisionworld.com/). (b) Electrical Conduction system of the heart

pumps low-oxygen blood to the lungs. The blood gets oxygenated in the lungs and gathers into the left ventricle. In the stemic circulation, the oxygenated blood in the left atrium is pumped into the left ventricle. The left ventricle pumps the blood to the rest of the body and the heart itself. After the body extracts the oxygen from the blood and injects carbon dioxide, the oxygen-depleted blood then flows back to the right atrium.

2.1.2 Electrical Conduction System of the Heart

The oxygen demand of the body changes during different activities. For example, the demand is higher while running and lower while sleeping. To satisfy these demands, the heart muscles in the atria and the ventricles have to contract with certain pattern and frequency in accordance to optimize the *Cardiac Output*, which refers to the volume of blood pumped by the heart per minute (mL blood/min). The coordinated contractions of the heart muscles are governed by the electrical conduction system of the heart (Fig. 2.1.(b)) A *Normal Sinus Rhythm (NSR)* is the healthy heart rhythm which provides efficient blood flow. During a NSR, electrical signals are periodically generated by the *Sinoatrial (SA) node* in the upper right atrium, which acts as the intrinsic pacemaker of the heart. The signals conduct throughout both atria and trigger

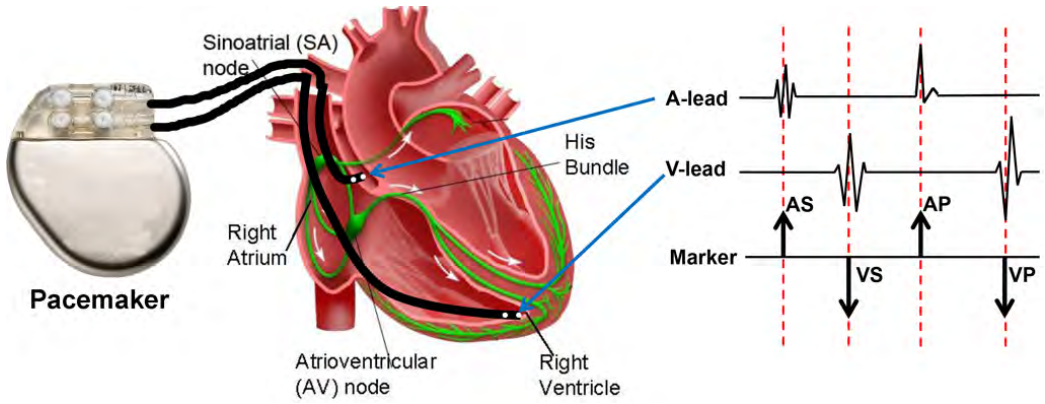


Figure 2.2: (a) Lead placement for a dual chamber pacemaker. (b) Electrogram (EGM) signals measured from pacemaker leads and corresponding internal pacemaker events

muscle contractions to push blood into the ventricles. After a long conduction delay at the *AV node* so that both ventricles are fully filled, the signals conduct through fast-conducting *His-Purkinje* system to trigger almost simultaneous contractions of the ventricles and pump blood out of the ventricles.

Derangement from NSR can result in insufficient cardiac output and thus insufficient oxygen supply to the body and/or the heart itself, which are referred to as *Arrythmia*. Arrhythmia impair the heart's ability to efficiently pump blood and compromise the patient's health. Arrhythmia are categorized into so-called *Tachycardia* and *Bradycardia*. Tachycardia features undesirable fast heart rate which can cause inefficient blood pumping. Bradycardia features slow heart rate which results in insufficient blood supply. Bradycardia are due to failure of impulse generation with anomalies in the SA node, or failure of impulse propagation where the conduction from atria to the ventricles is delayed or blocked.

2.1.3 Electrophysiology and Implantable Cardiac Devices

The electrical activities of the heart closely couple with the mechanical contractions thus the electrical activities of the heart can be monitored and used to diagnose arrhythmia. The most well-known method is Electrocardiogram (ECG), which measures the integration of electrical activities of the heart measured along different axis on the body surface. The electrical activities can also be directly measured by inserting electrodes through the vein into the heart. The electrodes are placed against the inside heart wall and localized electrical activities can be measured. Physicians can also deliver pacing sequence through the electrodes to explore the heart conditions. This procedure is referred to as Electrophysiological (EP) Testing (Josephson [2008]) and the signals are referred to as electrograms (EGMs) (Fig. 2.2.b). The timing and morphology of the ECG and EGM signals together are used to diagnose arrhythmia.

The implantable cardiac pacemakers are rhythm management devices designed to

treat bradycardia. A typical dual chamber pacemaker has two leads inserted into the heart through the veins which can measure the local electrical activity of the right atrium and right ventricle respectively (Fig. 2.2.a). According to the timing between sensed impulses, the pacemaker may deliver electrical pacing to the corresponding chamber to maintain proper heart rhythm.

2.2 A Dual Chamber Pacemaker Specification

The focus of this section is implantable pacemaker, which is one of the simpler implantable cardiac devices. The specifications are based on the algorithm descriptions from Boston Scientific manuals (Boston Scientific Corporation [2007b]) and the functional description released as part of the Pacemaker Challenge (Boston Scientific Corporation [2007a]).

The pacemaker is designed for patients with bradycardia (i.e. slow heart rate). Two leads, one in the right atrium and one in the right ventricle, are inserted into the heart and fixed onto the inner wall of the heart. These two leads monitors the local activation of the atria and the ventricles, and generate corresponding sensed events (AS, VS) to its software. The software determines the heart condition by measuring time difference between events and delivers pacing events (AP, VP) to the analog circuit when necessary. The analog circuit then delivers pacing signals to the heart to maintain heart rate and A-V synchrony. In order to deal with different heart condition, pacemakers are able to operate in different modes. The modes are labeled using a three character system (e.g. xyz). The first position describes the pacing locations, the second location describes the sensing locations, and the third position describes how the pacemaker software responds to sensing. Here we introduce the widely used DDD mode pacemaker which is a dual chamber mode with sensing and pacing in both atrium and ventricle.

A DDD pacemaker has five basic timing cycles triggered by external and internal events, as shown in Fig. 2.3.

Lower Rate Interval (LRI)

The Lower Rate Interval (LRI) defines the longest interval allowed between two ventricular events, thus keeping the heart rate above a minimum value. In DDD mode, the LRI interval is divided into a V-A interval (TLRI-TAVI) and a A-V interval (TAVI). Since the last ventricular event (VS, VP), if no atrial event has been sensed (AS), the pacemaker will deliver atrial pacing (AP) after TLRI-TAVI. (Marker 1 in Fig. 2.3)

Atrio-Ventricular Interval (AVI) and Upper Rate Interval (URI)

The function of the AVI timer defines the longest interval between an atrial event and a ventricular event. If there is no ventricular event (VS) within TAVI after an

atrial event (AS, AP), and the time since the last ventricular event (VS, VP) is longer than TURI, the pacemaker will deliver ventricular pacing (VP). (Marker 3 in Fig. 2.3) The URI limits the ventricular pacing rate by enforcing a lower bound on the times between consecutive ventricle events.

2.2.1 Post Ventricular Atrial Refractory Period (PVARP) and Post Ventricular Atrial Blanking (PVAB)

Ventricular events, especially Ventricular Pace (VP) are sometimes so strong that the atrial lead can sense the activation as well. This signal may be falsely recognized as an atrial event and disrupt normal pacemaker function. This scenario is called crosstalk and was discussed in our previous work (Jiang and Mangharam [2011]). In order to prevent this undesired behavior, and filter potential noises, there is a blanking period (PVAB) followed by a refractory period (PVARP) for the atrial events after each ventricular event (VS, VP). Atrial events during PVAB are ignored and atrial events during PVARP trigger AR! events which can be used in some advanced diagnostic algorithms. (Marker 2 in Fig. 2.3)

2.2.2 Ventricular Refractory Period (VRP)

The VRP follows each ventricular event (VP, VS) to filter noise and early events in the ventricular channel which could otherwise cause undesired pacemaker behavior.

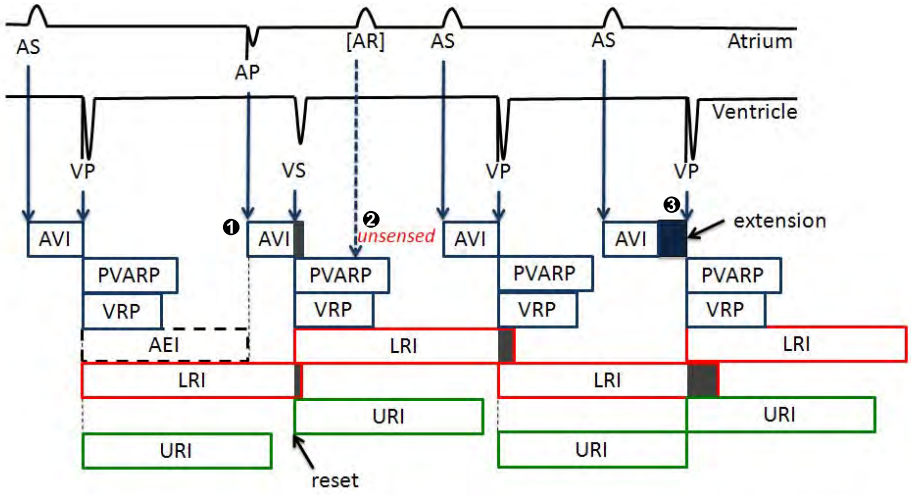


Figure 2.3: Basic 5 timing cycles for a dual chamber pacemaker which include the Lower Rate Interval (LRI), Atrio-Ventricular Interval (AVI), and Upper Rate Interval (URI). Also included are the blanking intervals, Post Ventricular Atrial Refractory Period (PVARP) and Ventricular Refractory Period (VRP), to inhabit action by the pacemaker.

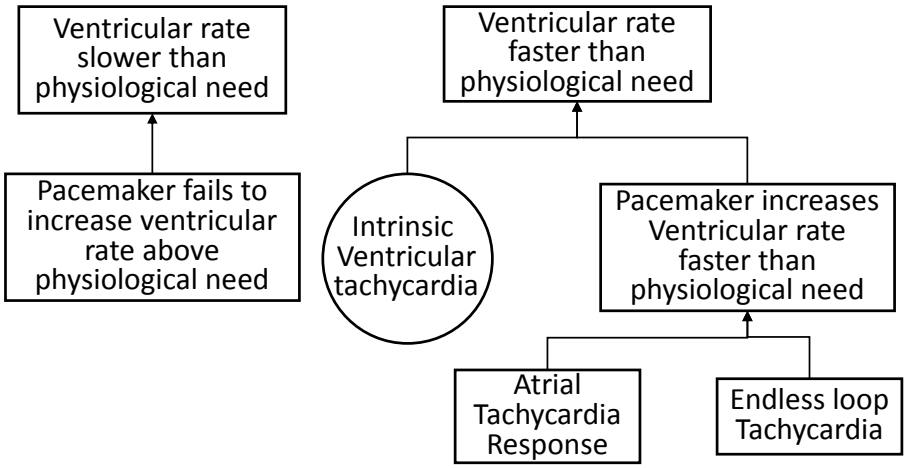


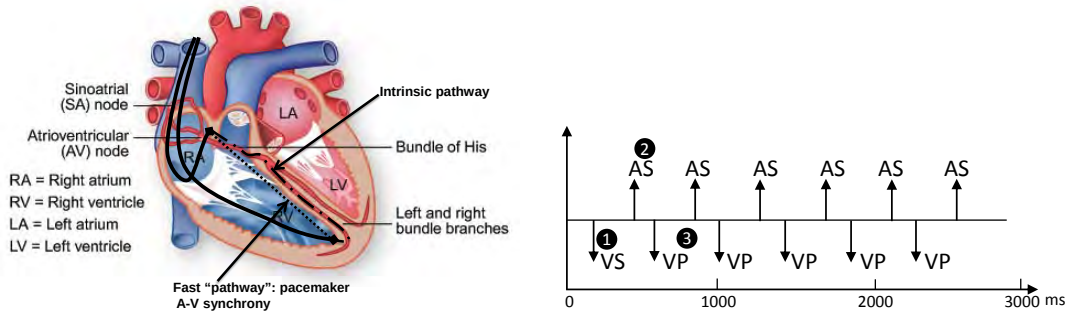
Figure 2.4: Fault Tree Analysis (FTA) for two failures of a pacemaker

2.3 Identify Hazards in the Dual Chamber Pacemaker Design

Implantable pacemakers are designed to treat bradycardia by increasing the heart rate with external pacing. Therefore the heart rate should not only be 1) increased to the minimum physiological need, but also 2) should not be increased beyond physiological need. Failing to satisfy the two requirements leads to failures that may be harmful to the patient. Fault tree analysis (FTA) is a top down and deductive failure analysis in which an undesired state of a system is analyzed using Boolean logic to combine a series of lower-level events. Fig. 2.4 demonstrates two FTAs for two failures corresponding to the two requirements. In this section we introduce two well-studied safety hazards in a basic dual chamber pacemaker design.

2.3.1 Endless-Loop Tachycardia

As introduced in the last section, a dual-chamber pacemaker paces the ventricle if no ventricular events are sensed after TAVI, which is equivalent to a virtual atria-to-ventricles conduction pathway. This forms a timing loop with the intrinsic (physiological) A-V conduction pathway (see Fig. 2.5(a)). A Premature Ventricular Contraction (PVC), i.e. an early extra beat in the ventricles, may trigger another ventricular event (VS) and initiate a V-A conduction through the intrinsic pathway (Marker 1 in Fig. 2.5(b)). The pacemaker registers this signal as an Atrial Sense (AS) (Marker 2 in Fig. 2.5(b)). A ventricular pace (VP) is delivered after TAVI, as if the signal conducts through the "virtual" A-V pathway (Marker 3 in Fig. 2.5(b)). The VP will trigger another V-A conduction and this VP-AS-VP-AS looping behavior will continue (see Fig. 2.5(b)). The interval between atrial events is TAVI plus the V-A conduction delay, which is normally shorter than the delay between intrinsic heart beats, thus



(a) Virtual circuit formed by the pacemaker and the heart

(b) Pacemaker trace for ELT initialized by a early ventricular signal

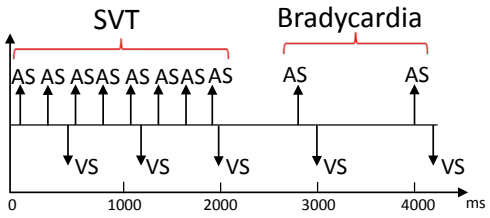
Figure 2.5: Endless Loop Tachycardia case study demonstrating the situation when the pacemaker drives the heart into an unsafe state (Jiang et al. [2011])

driving the ventricular rate as high as the Upper Rate Limit. During ELT, the heart rate is not only high, but also fixed without changing according to physiological need, which is an unsafe scenario.

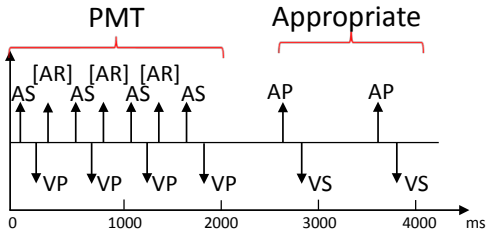
2.3.2 Atrial Tachycardia Response

Supraventricular Tachycardia (SVT) is an arrhythmia with an abnormally fast atrial rate. Typically, in a heart without pacemaker, the AV node, which has a long refractory period, can filter most of the fast atrial activations during SVT, thus the ventricular rate remains relatively normal. Fig. 2.6(a) demonstrates a pacemaker event trace during SVT, with a pacemaker in ODO mode, which just sensing in both channels. As there is no pacing in ODO mode, the heart is in open-loop with the pacemaker. In this particular case, every 3 atrial events (AS) correspond to 1 ventricular event (VS) during SVT. As an arrhythmia, SVT is still considered a safe heart condition since the ventricles operate under normal rate and still maintain adequate cardiac output.

However, in the closed loop case with the DDD pacemaker, the AVI component of a dual chamber pacemaker is equivalent to a virtual pathway in parallel to the intrinsic conduction pathway between the atria and the ventricles. The pacemaker tries to maintain 1:1 A-V conduction and thus increases the ventricular rate inappropriately to match the atrial rate. This is known as Pacemaker Mediated Tachycardia (PMT) as the heart would have been safe without the pacemaker and its virtual pathway. Fig. 2.6(b) shows the pacemaker trace of the same SVT case with DDD pacemaker. Although half of the fast atrial events are filtered by the PVARP period ([AR]s), the DDD pacemaker still drives the closed-loop system into 2:1 A-V conduction with faster ventricular rate. Maintaining A-V delay is less important than maintaining an appropriate ventricular rate. The DDD pacemaker violates a higher priority requirement in order to satisfy a lower priority requirement, which is inappropriate.



(a)



(b)

Figure 2.6: Benign open loop case: SVT without a pacemaker or with a pacemaker in sense-only mode (ODO) (b) Dangerous closed-loop-case SVT with DDD pacemaker which tries to match the fast atrial rate with a corresponding (and dangerous) fast ventricular rate.

2.4 Discussion

Implantable cardiac devices such as implantable pacemakers are typical autonomous medical devices. Despite their seemingly simple controllers, the pacemakers also subject to the three challenges discussed in Chapter 1:

- The physiology of the heart and its interaction with the rest of the body are complex.
- The pacemakers have to safely operate within a large variety of physiological conditions.
- A dual chamber pacemaker can only observe electrical activities from two local sites in the heart.

In the remaining dissertation, I will demonstrate the application of physiological models in various model-based techniques to provide safety and efficacy confidence to the pacemaker design.

Chapter 3

Theme 1: Modeling the Physiological Environment

The safety and efficacy of autonomous medical devices have to be evaluated within their physiological environment. Models of human physiology can replace real patients and enable closed-loop evaluation earlier during device development. In this chapter, a heart model structure is developed for closed-loop validation of implantable cardiac devices. This chapter aim to address the following questions:

- How much detail does the physiological model need?
- How to validate the physiological model?
- What applications can the physiological models be used?

3.1 Related Work

To study the mechanisms of heart diseases and their effects on cardiac output, different physiological models of the heart have been developed. Fig. 3.1 illustrates several aspects that these models capture. With the development of the imaging techniques like MRI, detailed anatomical structures of the heart can be modeled and studied (Schulte et al. [2001]). These models are fundamental in other modeling aspects as well, as the anatomy of the heart dictates the electrical and mechanical behaviors of the heart. Fig. 3.1.(a) shows models for heart muscle fiber orientations by E.W. Hsu and C.S. Henriquez [2011]. With anatomy models the electrical and/or mechanical properties of the heart can be studied. Fig. 3.1.(b) illustrate a model of blood flow within the ventricles (Peskin and McQueen [1989]). Electrical properties of the heart at cellular level has been modeled (Sachse et al. [2008]) and by combining these cellular models with the structural models, the electrical activities of the whole heart are studied, especially the mechanism of different arrhythmia (Trayanova and Boyle [2014], Grosu et al. [2011], Murthy et al. [2013]). Intrinsic heart rate variability has

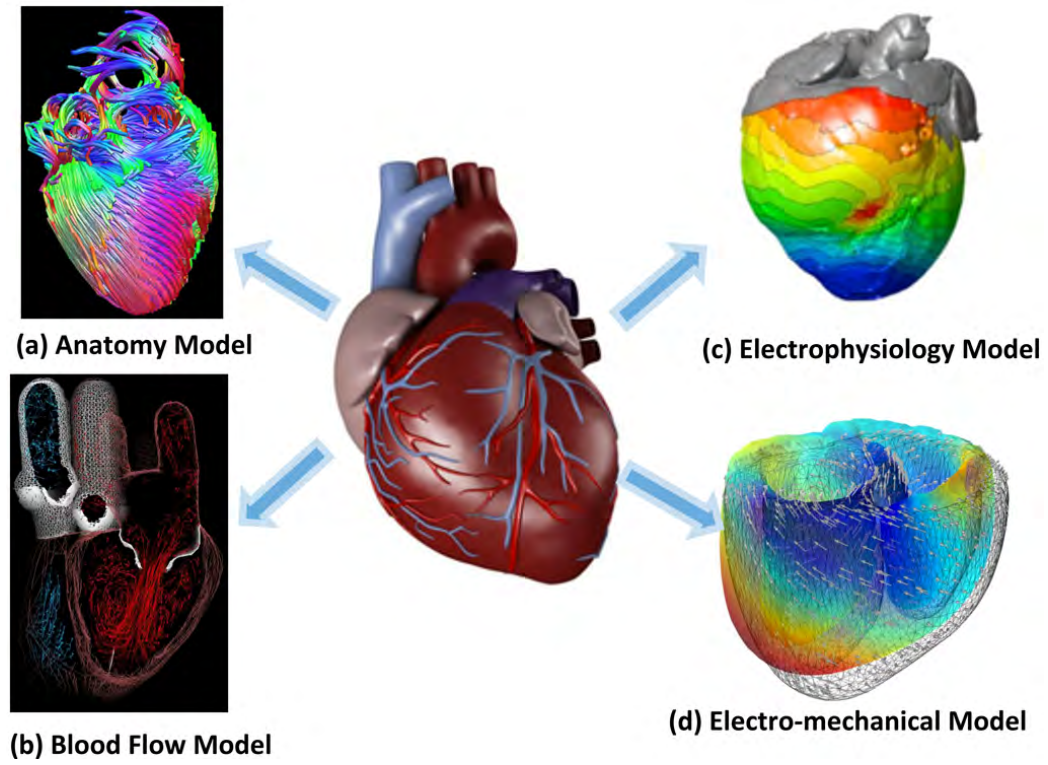


Figure 3.1: Physiological models of the heart from different perspectives

been modeled to synthesize optimal control of pacemaker pacing. (Bogdan et al. [2013]) Abstraction of the electrical cellular model has also been attempted by Islam et al. [2014] to reduce model complexity without sacrificing accuracy. The electrical properties and the mechanical properties of the heart are closely coupled. Models combining both of these aspects are also developed to study the effects of different arrhythmia on cardiac outputs (Trayanova and Boyle [2014], Rossi et al. [2011]).

3.2 EP Heart Model Structure for Closed-loop Validation of Implant-able Cardiac Devices

Models should be designed in accordance with their respective applications. The aforementioned models of the heart are designed for understanding the mechanisms of different heart diseases. For closed-loop evaluation of autonomous medical devices, physiological modeling should have the following considerations:

C1. Interfacing with the device: The model should be able to generate physiological signals that the device sense from the real physiological entities. The model should also be able to take device output as input and change its states accordingly. Model complexity should also be adjusted according to the device interface to hide unnecessary details.

C2. Differentiate different physiological conditions: To evaluate the safety and effectiveness of the device, the device has to be evaluated under certain physiological conditions specified by the requirements. For example, the pacemaker is supposed to maintain proper heart rate during Bradycardia. The model should be expressive enough to be able to differentiate the physiological condition (Bradycardia in the example) from other conditions. Failing to do so may result in false-positives or false-negatives in the evaluation result.

C3. Physiological/logical interpretation of model states: In closed-loop evaluation we are checking the device safety and effectiveness against the physiological requirements. However, due to the limited interface (e.g. two leads for a dual chamber pacemaker) it is always difficult to determine only from an execution trace that the therapy is safe and effective. Therefore, being able to provide physiological meanings to the states of the model also allows us to interpret the closed-loop execution more accurately, thus reducing the number of physiologically invalid executions during the evaluation. To satisfy these requirements, the model structure of these physiological models should base on physiological or clinical first principles so that states and state transitions of the closed-loop executions can be explained with physiological language.

C4. Available patient data: In closed-loop evaluation, physiological models are developed to represent certain physiological condition across a population of patients or even particular patients. The model parameters must be identified so that the behaviors of the models match the behaviors of the patients (groups). Due to the limited sensing capability of closed-loop medical devices, the obtained data is sparse: i.e. we can not put a sensor on every tissue region of the heart. Therefore the complexity of the model should be in accordance with the available data to avoid *over-fitting*, which occurs when a model has too many parameters relative to the number of observations, and this can introduce errors during prediction.

The electrophysiological models mentioned in the last section (Trayanova and Boyle [2014], Grosu et al. [2011]) satisfy C1-C3. However, the parameter space of these models are too large (10+ parameters for each cellular model multiplied by 10^5 of elements) which not only increase simulation complexity, but also impossible to identify due to lack of data. As introduced in Section 2.1.3, the pacemaker has only two leads at fixed locations and only use timing between local activation events for diagnosis. These models with high spatial fidelity possess details that can be abstracted without sacrificing the model accuracy.

Electrophysiology testing (EP testing) has been an active clinical field to diagnose and treat arrhythmia with minimal-invasive procedures. During an EP testing procedure, the physicians diagnose heart conditions by examining the patterns and intervals of local electrical activations (temporal) measured from electrodes placed into different locations of the heart (spatial). EP testing is the perfect level of abstraction for closed-loop evaluation of implantable cardiac devices because: 1) it is the basis of implantable cardiac devices (C1), 2) physicians can use EP testing to

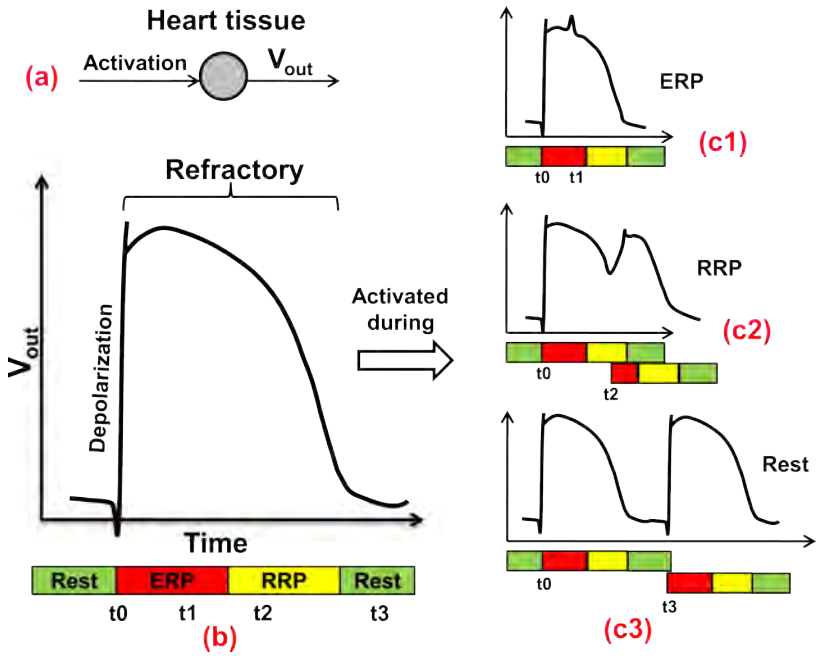


Figure 3.2: (a) The generation of Action potential; (b) Action potential; (c1) The second activation arrived during ERP; (c2) Arrived during RRP; (c3) Arrived after refractory.

diagnose most arrhythmia thus distinguish them (C2,C3), and 3) there are abundant patient data available (C4).

In the remaining chapter we will introduce the heart model structure based on EP testing, and model adaptation for two different applications of closed-loop evaluation of implantable cardiac devices.

3.2.1 Timing Behaviors of Cellular Electrophysiology

The contraction of heart muscles is triggered by external voltage applied to the tissue. After the activation, a transmembrane voltage change over time can be sensed due to ion channel activities, which is referred to as an Action Potential (Fig. 3.2(a)). The upstroke of the action potential is called depolarization, during which the muscle will contract. The voltage change caused by the depolarization will depolarize the tissue nearby, which causes an activation wave across the heart. After the depolarization there is a refractory period during which the tissue recovers to the pre-excitation state and the voltage drops down to the resting potential. The refractory period can be divided into *Effective Refractory Period (ERP)* and *Relative Refractory Period (RRP)* (Fig. 3.2(b)). During ERP, the tissue cannot be depolarized due to the lack of charge. As a result, the activation wave will be "blocked" at the tissue during ERP (Fig. 3.2(c1)). During RRP, the tissue is partially recovered and the tissue can be depolarized. However, the new action potential generated by the depolarization will have different morphology (e.g. attenuated in magnitude and duration), thus affecting the refractory periods of the tissue and conduction delay of the activation

wave (Fig. 3.2(c2)). Fig. 3.2(c1)-(c3) show the action potential shape change and corresponding timing change in refractory periods when the tissue is activated at time stamp $t1$, $t2$, $t3$ after the initial activation $t0$.

3.2.2 Heart Model Components

We introduce the model components that can be used to configure heart models corresponding to different heart conditions. As discussed earlier, the action potential of a heart tissue has 3 timing periods during which the tissue responds to external electrical stimuli differently. We use an extended timed-automata formulation (Alur and Dill [1994]) to model the timing behaviors of a heart tissue during each cycle.

Node Automata: We refer to the tissue model as *node automaton* and Fig. 3.3.(a) shows the structure of a node automaton i. 3 states correspond to the timing periods of the action potential. From Rest state, the node can either self-activate or get activated by external stimuli (Act_node) and go to ERP state. During ERP state the node does not respond to external stimuli (blocked). During RRP state, the node can still be activated and go to ERP state, however the ERP period and the conduction delay of the tissue are affected by the "earliness" of the activation arrived during the RRP period, which is tracked by a shared variable $C(i)$. The new ERP period is determined by a function over clock value $g(f(t))$ which mimics the beat-to-beat dynamics described in Josephson [2008]. The function g and f are given by:

$$f(t) = 1 - t/Trrp \tag{3.1}$$

and

$$g(x) = \begin{cases} T_{min} + (1 - (1-x)^3) \cdot (T_{max} - T_{min}), i = AV \\ T_{min} + (1 - x^3) \cdot (T_{max} - T_{min}), i \neq AV \end{cases} \tag{3.2}$$

where T_{min} and T_{max} are the minimum and maximum value for *Terp* of the tissue.

Due to the limited number of observable points within the heart, modeling the electrophysiological behavior of every tissue of the heart and its full anatomy is unnecessary and unfeasible. In our heart models, only self-activating tissue and key hubs of the electrical conduction system are modeled as node automata.

Path Automata: The electrical conduction through the tissue between nodes are abstracted using *path automata*. The path automata can be used to represent structural or topological (functional) electrical connections between nodes. Fig. 3.3.(b) shows a path automaton connecting node a and b.

The initial state of a path automaton is *Idle*, which corresponds to no conduction. The states corresponding to the two conduction directions are named after the physiological terms: Antegrade (Ante) and Retrograde (Retro). These states can be intuitively described as forward and backward conductions. If path actuation *Act_path* event is received from one of the nodes connected to it, there is a transition to *Ante* or *Retro* state based on the activation source in the path automaton. At the same time, the clock invariant of the state is modified according to the shared variable *C(a/b)*.

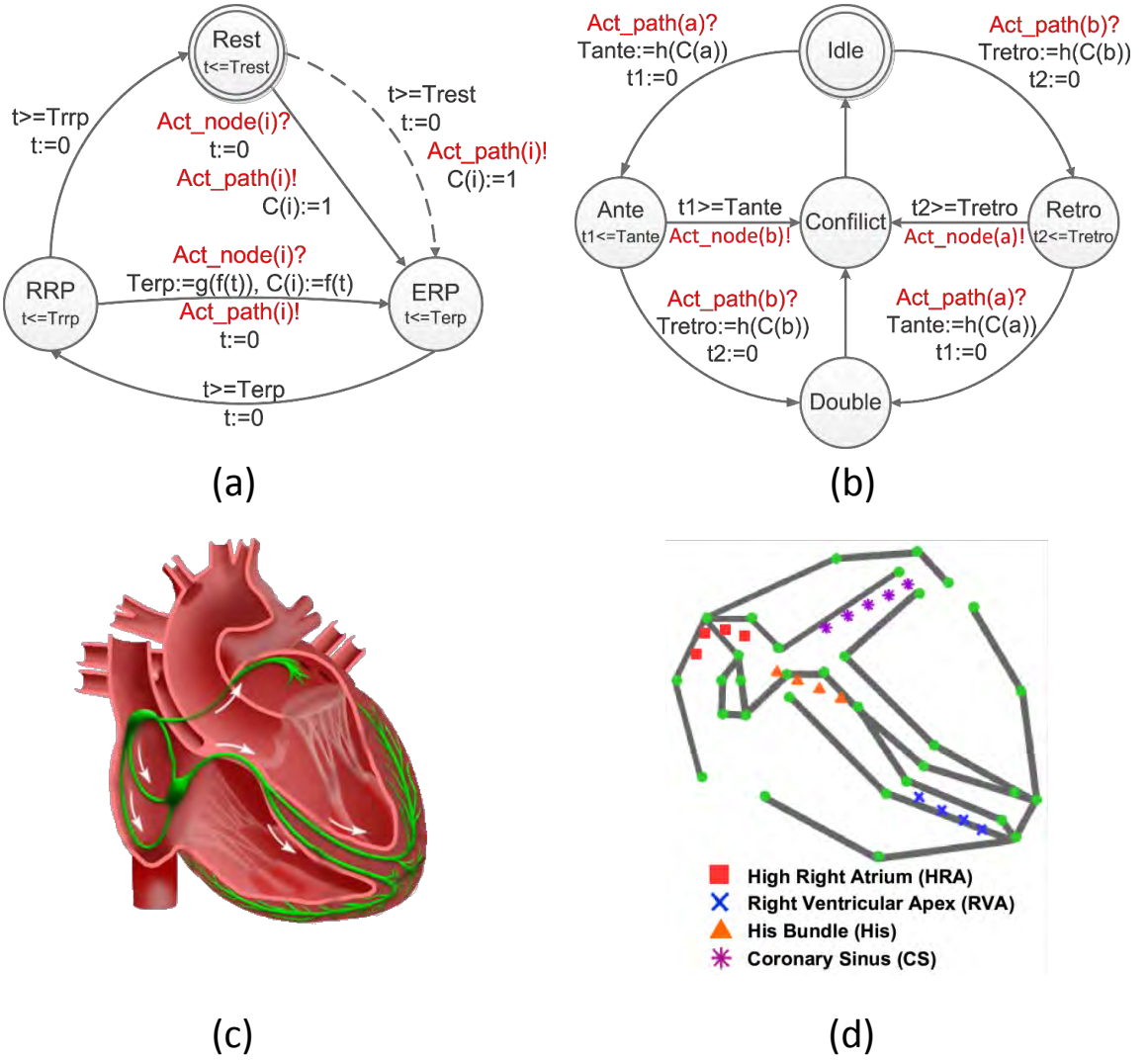


Figure 3.3: (a) Node automaton: The dotted transition is only valid for tissue (like SA node) that can be activated by an external trigger; (b) Path automaton modeling the electric conduction and propagation between two node automata; (c) Electrical conduction system of the heart; (d) Model of the electrical conduction system of the heart using a network of node & path automata (Jiang et al. [2012a]).

This corresponds to the change of the conduction delay that is caused by the early activation. Similar to node automaton, the changing trend is extracted from clinical data and the function h is defined as:

$$h(c) = \begin{cases} path\_len/v \cdot (1+3c), i = AV \\ path\_len/v \cdot (1+3c^2), i \neq AV \end{cases} \tag{3.3}$$

where *path_len* denotes the length of the path and v is the conduction velocity.

After *Tante* or *Tretro* time expires, the path automaton sends out *Act_node(b)* or *Act_node(a)* respectively. A transition to *Conflict* state occurs followed by the

transition to *Idle* state. The intermediate state *Conflict* is designed to prevent backflow, where the path is activated by the node b it has just activated. If during *Ante* or *Retro* state another *Act_path* event is received from the other node connected to the path automaton, a transition to *Double* state will occur, corresponding to the two-way conduction. In this case, the activation signals eventually cancel each other and the transition to *Idle* state is taken.

3.2.3 Modeling the Heart's Electrical Conduction System

The node and path automata are the basic building blocks for EP heart modeling. Hearts with different conditions are modeled by using different conduction topologies with appropriate timing parameters for each node and path automata. Fig. 3.3.(d) shows one such topology of a network of node and path automata.

3.3 Interaction with the Heart Model

In this section, we first introduce a probe model we developed to generate synthetic EGM signals from the EP heart model. We then use two case study to demonstrate that the probe model enables the EP heart model to evaluate device malfunctions due to sensing errors.

3.3.1 Probe Model for Synthetic EGM Generation

In EP testing and during pacemaker implantation, the local electrical activities, measured as electrogram (EGM) signals, are used to diagnose heart conditions. During heart model construction, we can assign a node automaton at electrode locations and the transitions to the ERP state can be used to represent the local activation events. In a more general setup where electrodes are assigned anywhere within the heart model, a probe model is designed to generate synthetic EGM signals using spatio-temporal information from the proximity to the network of node and path automata.

According to Stevenson and Soejima [2005], a potential difference is generated when the activation wavefront passes by the electrode. The locations of the activation wavefronts are calculated from the locations of the path automata and their current timer values. The amplitude of EGM decreases when the activation wavefront moves away from the probe. We assume the decrease factor is a function related to the distance between the activation wavefront and the probe. The potential difference caused by an activation wavefront to a probe is the signal strength of the path multiplied by the decrease factor. The amplitude of EGM from a probe is the sum of potential differences caused by all activation wavefronts. The bipolar EGM is the difference between two unipolar EGMs. Fig. 3.4 shows that this probe model captures timing properties of EGM and the functional shape of the EGM impulses. The

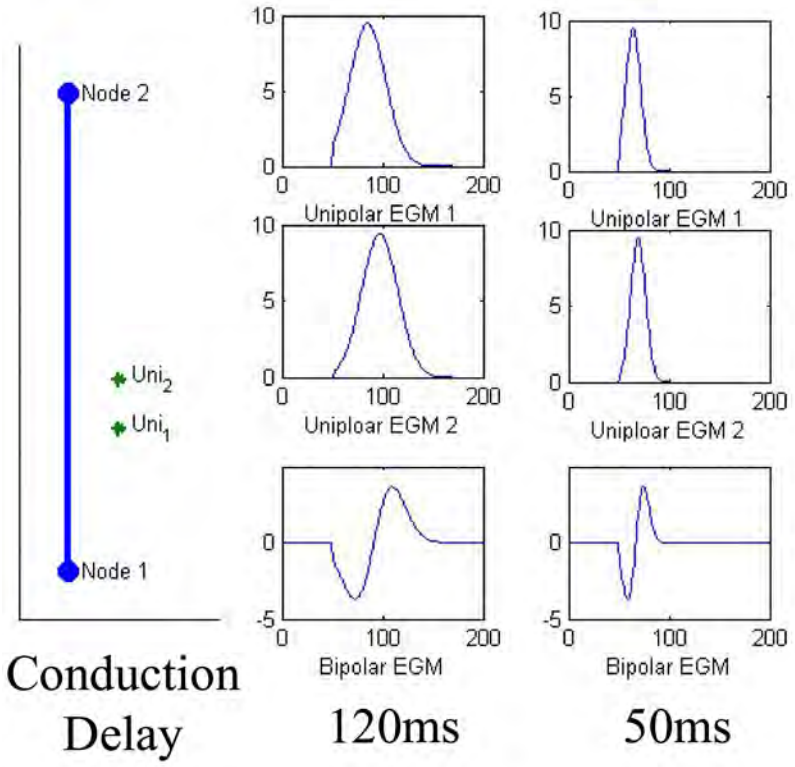


Figure 3.4: The influence of conduction velocity and probe configuration on the EGM morphology. The left columns show the placement of probes in relation to the path; the right columns show the functional EGM.

probes can be placed anywhere within the heart model and generate clinically-relevant EGMs.

With the sensing model, the heart model structure can be used to identify safety hazards caused by sensing errors.

3.3.2 Pacemaker Oversensing and Crosstalk

Oversensing is a general term for inappropriate sensing caused by noise or far-field signals. It's very common among pacemaker malfunctions and it may result in failure to pace (Beaumont et al. [1995], Fuertes and Toquero [2003]), competitive pacing and inappropriate therapy. Crosstalk is a special case for oversensing which occurs when the pacemaker stimulus in one chamber is sensed in the other chamber. It happens when two leads are close to each other or pacing signal in the other chamber is too strong. It is common that the ventricular lead is placed in the right ventricle outflow tract, which is close to the atrium (Saxonhouse et al. [2005]). Fig. 3.5(a) shows simulated EGMs from a patient with bradycardia and complete heart block. During atrial pacing (AP), the pacing signal is sensed by the ventricular lead 53 ms after the AP. (Marker 1) It is treated as ventricular sense (VS) signal and thus inhibits the

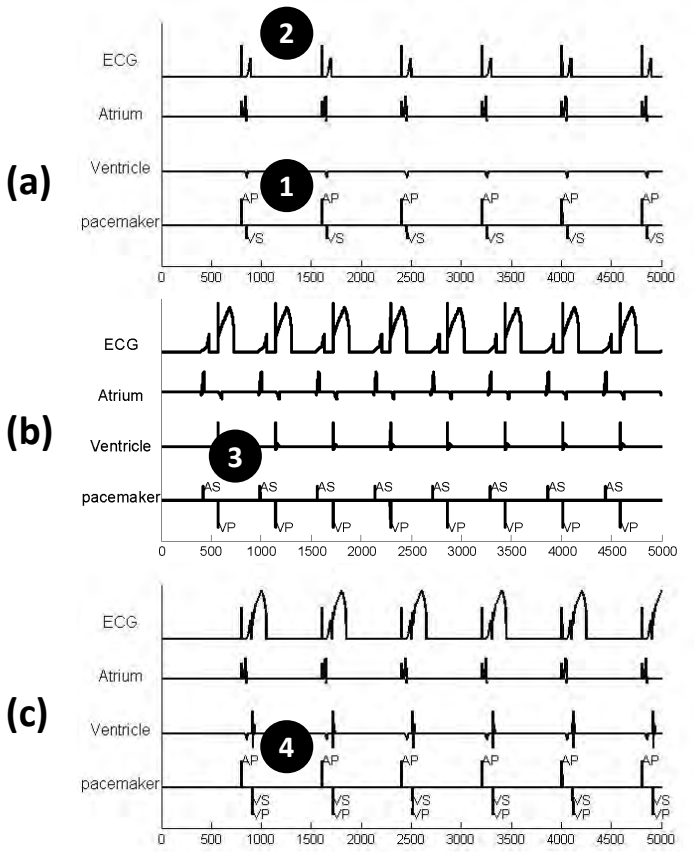


Figure 3.5: Crosstalk between pacemaker leads with high sensitivity in the ventricle, adjusted sensitivity and ventricular safety pacing

subsequent ventricular pacing (VP). This is indicated by no QRS-wave in the ECG channel. (Marker 2) For a patient with complete heart block this will cause dangerous ventricular asystole, meaning a long time without ventricular events.

Increasing the sensing threshold of the ventricular channel can prevent false sensing. In Fig. 3.5(b), the small signals in ventricular EGM are ignored and ventricular pacing are successfully delivered.

3.3.3 Lead Displacement

Lead displacement affects many patients and can result in inappropriate or ineffective therapy. Fig. 3.6. (b) shows the simulation result for the pacemaker function when the leads are in their designated location. From the figure we can observe: 1) Each P-wave is initialized by an Atrial Pace signal. 2) Each QRS complex is initiated by a ventricular pacing signal. 3) The interval between AP and VP is 150 ms, which matches the programed AVI period.

One common case for lead dislodge is shown in Fig. 3.6.(a), where the atrial lead has fallen into the right ventricle outflow tract. In this case the atrial lead senses from

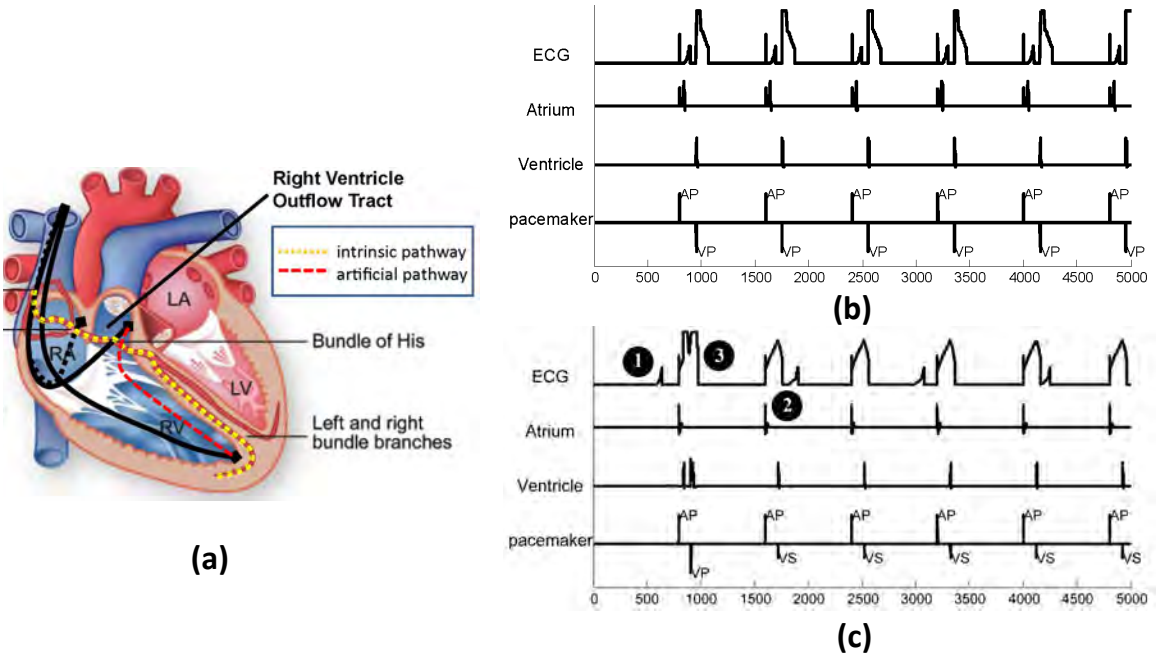


Figure 3.6: (a) Dotted line shows the location where the atrial lead should be (b) Pacemaker function before lead dislodge. (c) Pacemaker function after lead dislodge

the ventricle rather than atrium and atrial pacing will initiate a ventricular event. Fig. 3.6.(c) shows the simulated EGMs in this case. The figure reveals several facts: 1) No P wave is sensed or tracked (Marker 1). 2) Atrial Pace initiates an abnormal, wide QRS which is then sensed by the ventricle lead (Marker 2). 3) Intermittent appearance of VP on QRS 110 ms after the AP. The ventricular lead can receive signal from: 1) pacing signal sent from the atrial lead, 2) the intrinsic A-V conduction path. The two paths are shown in Fig. 3.6.(a) and form a timing race condition. When the signal from the atrial lead arrives the ventricular lead first, it will trigger VS. If the intrinsic signal arrives the ventricular lead during the VSP sensing window (defined in previous section), it will trigger VSP. Although the pacing is 'safe' because the pacing is early enough to avoid the vulnerable refractory period, the damage caused by pacing on depolarized tissue is currently a matter of much investigation.

3.4 Heart-on-a-Chip Platform

The heart model structure is also available on hardware platform (Fig. 3.7) for closed-loop testing of pacemaker implementations. Since each heart model is a network of node and path automata running concurrently, we implemented the heart model on an FPGA, so that increasing in the number of nodes and paths would not affect real-time constraints. The second generation heart model implementation has been

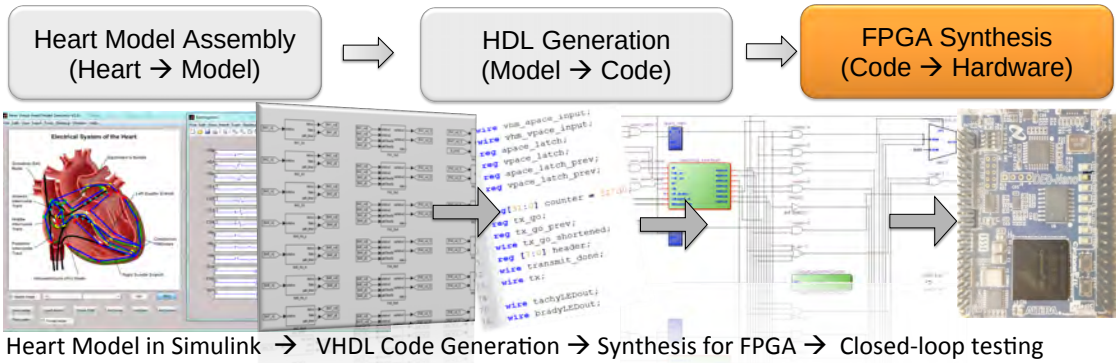


Figure 3.7: The heart model was developed in Matlab/Simulink and code was automatically generated to operate on an FPGA platform for platform-level testing.

implemented on a lower cost fast micro-controller platform. The fast clock ensures that executions of all nodes and paths can be finished within 1ms. The Heart-on-a-Chip platform includes a heart model implementation which is able to represent common heart conditions such as bradycardia, tachycardia, heart block, etc (for mode details refer to Jiang et al. [2012a]). The parameters of the heart model can be changed at run-time by either switching among pre-defined parameter sets, or sending values directly to the model through a user interface in Matlab. A monitoring system observes logical interactions between heart model and the pacemaker and checks them against safety invariants at run-time.

As shown in (Fig. 3.8), with an analog interface the heart model can interact with a commercial pacemaker in real time. Our analog interface uses an optical isolation circuit to separate the pacemaker circuit and the heart implementation. Signals generated from the heart are attenuated to the appropriate level to interact with a Boston Scientific pacemaker and analog pacing signals are converted to pacing events received by the heart model.

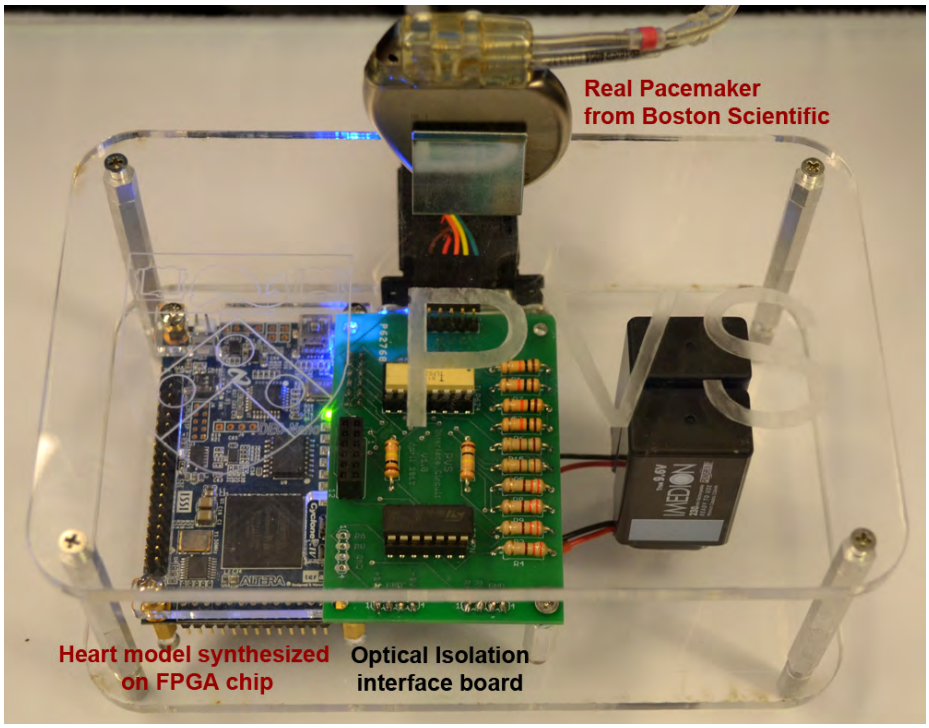


Figure 3.8: Heart-on-a-Chip testbed for real-time closed-loop testing of the pacemaker or model of the pacemaker with the heart model on the hardware platform

3.5 EP Heart Model Validation

Since models are approximations of the actual environment, there are always discrepancies between the model and the actual patient (group). The validity of closed-loop validation results directly correlated to the validity of the physiological models.

In this section we validate the heart model structure by demonstrating its capability to represent physiological behaviors of 1) a particular patient for closed-loop simulation, and 2) a group of patients for closed-loop model checking. The metrics and process to validate the heart model are different for the two applications of heart modeling: in closed-loop simulation, the **accuracy** of the model is more important, while in closed-loop model checking, the model's **coverage** on environmental behaviors is more important.

3.5.1 Validating Models for Closed-loop Simulation

A physiological model is considered valid for closed-loop simulation if (a) it is capable of generating the same output as the patient, for the same input; and (b) it is general enough to represent other patients with similar conditions by adjusting its parameters. The second point is to ensure that the model successfully captures the underlying mechanism instead of over-fitting the data. In the following example we validate the capability of our heart models to represent certain heart conditions according to the mechanisms described in physiological literature, and output the correct responses across a range of inputs.

Quantitative Heart Model Validation: During an EP testing procedure, the physician places catheters inside the patient's heart to observe local electrical activity from different locations of the heart. The His bundle catheter (HBE) is particularly important when evaluating the atria-to-ventricle conduction path (Fig. 3.9). For each A to V conduction there are 3 impulses which correspond to atrial contraction (A), His bundle activation (H) and ventricular activation (V). In this case study, two pacing signals $a1$ and $a2$ are delivered to the heart from the high right atrial catheter (HRA). By gradually decreasing the pacing interval in each test, certain tissue along the A-V conduction path will be activated during its refractory period, thus affecting the conduction delay further down the conduction path and change the intervals between the impulses. Fig. 3.10(a) shows the relation between pacing interval ($a1$-$a2$) and corresponding intervals between A, H and V impulses. On the left side it shows that interval $H_1 - H_2$ and $V_1 - V_2$ decrease but remain equal as the pacing interval decreases, indicating the tissue with the longest refractory period along the path is not between the His Bundle and the ventricles. When the pacing interval decreases to 350ms both intervals increases, indicating that the RRP of certain tissue has been reached and the tissue is between the atria and the His bundle. On the right it shows that the $A_2 - H_2$ interval increases as the pacing interval decreases, which further proves the hypothesis that the AV node, which is between the atria and the His bundle, has the

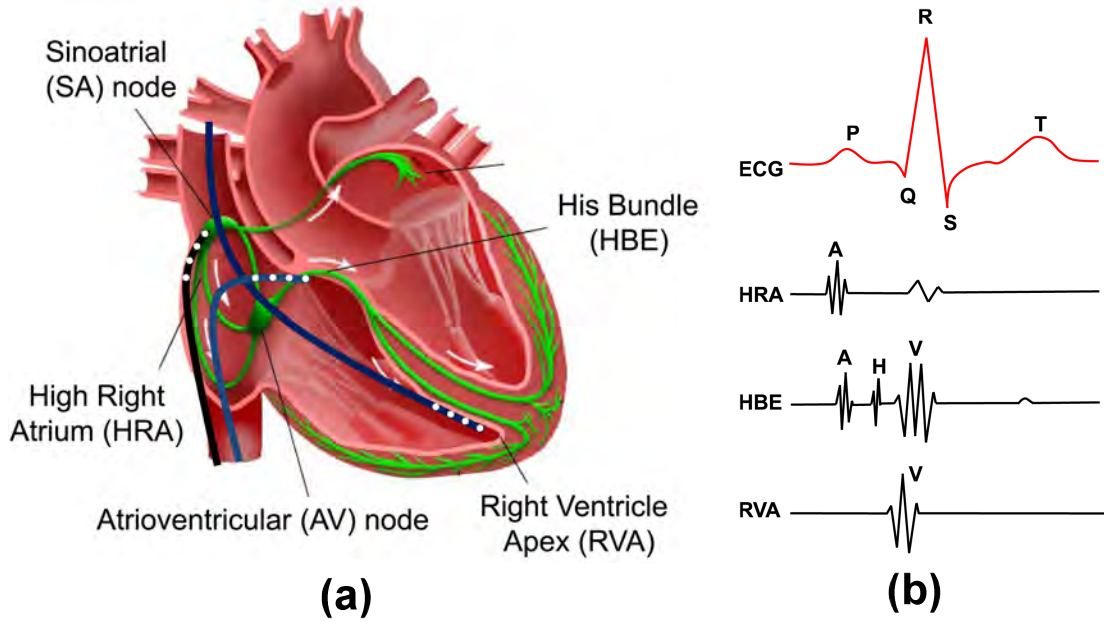


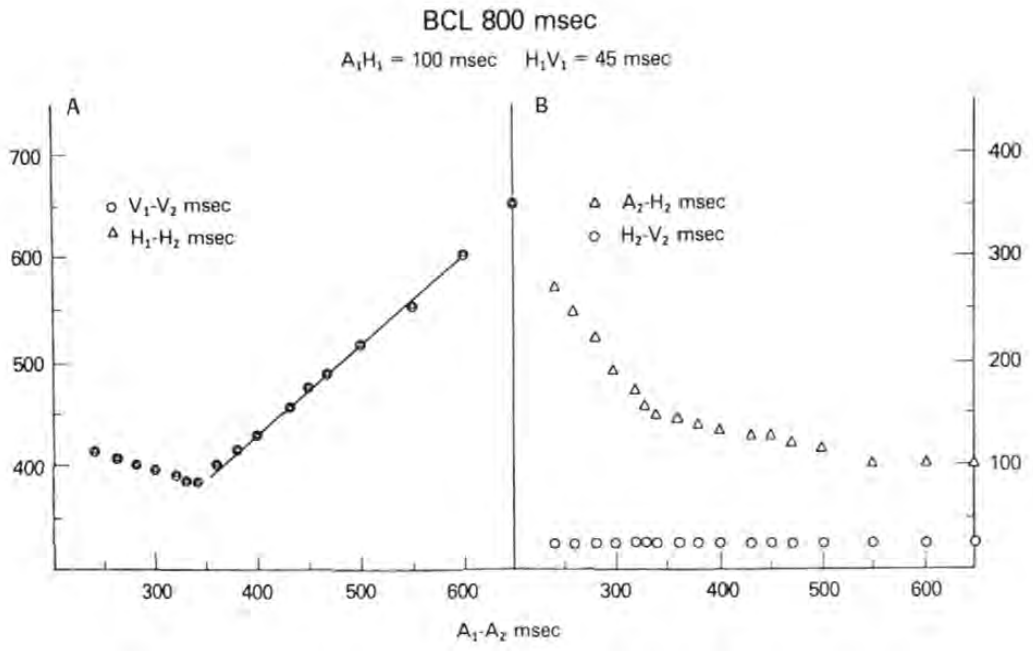
Figure 3.9: (a) Probe locations for a general EP testing procedure. (b) EGM signals measured from the probes at the high right atrial (HRA), His bundle (HBE) and right ventricular apex (RVA) standard catheter positions

longest refractory period along the A-V conduction path. We configured our heart model such that the AV node has the longest refractory period and performed the same study by decreasing the pacing interval. The heart model shows the same trend as that of the real patient (Fig. 3.10(b)).

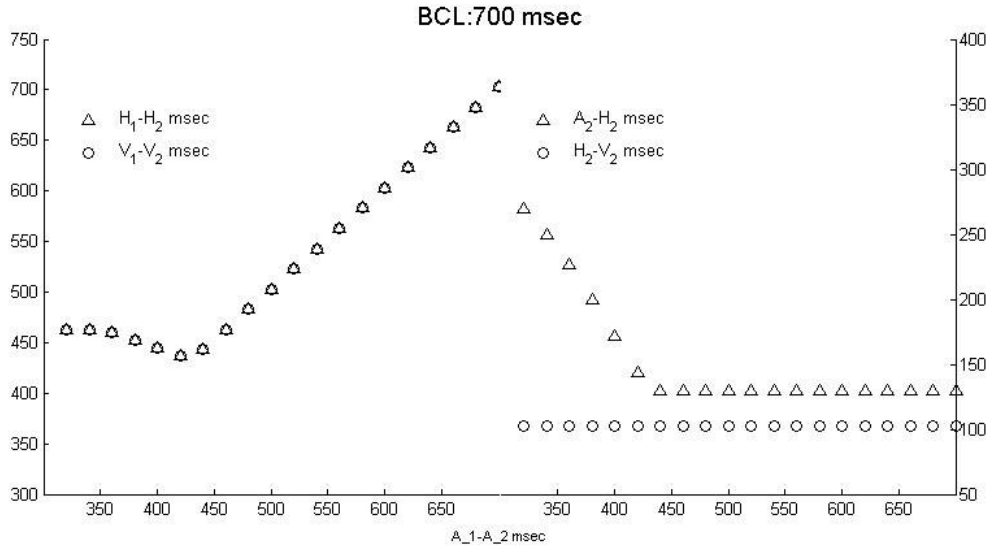
Validation by comparison to real patients: This heart condition can also show Wenckebach type A-V nodal response. In this case, a sequence of pacing signals with a short coupling interval ($A_1 - A_2 <= AV.Terp + AV.Trrp$) is delivered in the atrium. This results in a gradual increase in the AV nodal conduction delay and then a dropped beat occurs in the ventricle due to the increased ERP period of the AV node. The EGMs for a real patient with Wenckebach type A-V nodal response are shown in Fig. 3.11(a). With the VHM, we observe similar behavior, and the gradually increasing ERP and conduction delay are visualized in Fig. 3.11(b).

3.5.2 Validating Models for Closed-loop Model Checking

In model checking, a lot of complex dynamics of the environment are abstracted so that the environment model covers a larger number of environmental behaviors using non-determinism. The validity of the model is obtained by a valid initial model and a rigorous abstraction processes. In Jiang et al. [2014], we started with a valid detailed deterministic model (as described above) and by applying different abstraction steps we were able to generate a series of non-deterministic heart models. Between each abstraction step, the heart models satisfy a timed simulation relationship (Yamane



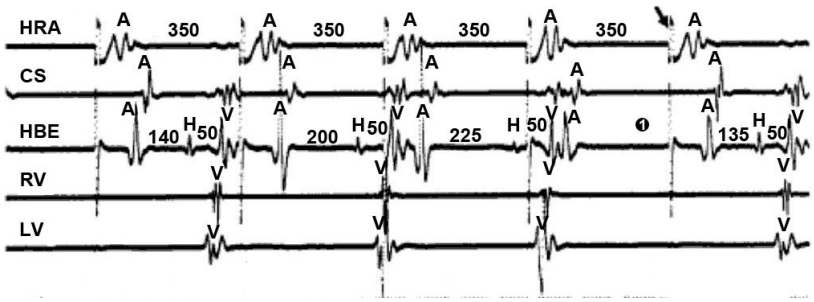
(a)



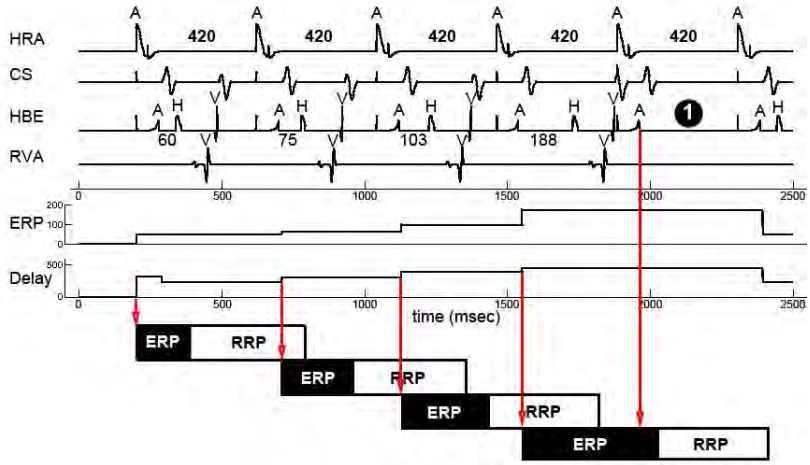
(b)

Figure 3.10: Key interval values when the coupling interval shortens for (a) a real patient (Josephson [2008]) and (b) in heart model simulation (Jiang et al. [2010b]).

[2006]) which is described below. The timed simulation guarantees all behaviors are covered in the more abstract model. More details regarding heart model abstraction will be discussed in Chapter 3.



(a) Real patient's electrograms



(b) Heart model's electrograms

Figure 3.11: (a) Electrograms of induced Wenckebach block in a patient. (b) Electrograms of induced Wenckebach block in the heart model with a basic cycle length of 420 msec. The heart model also displays lengthening in the A-H interval and block in A-V node (Marker 1). Rows 5 and 6 show the increase in the ERP and conduction delay of the A-V node.

3.6 EP Heart Model Identification

Physiological models are developed to represent certain clinical conditions common across a population of patients, or the conditions of a specific patient. Consequently, the structure of the model and corresponding parameters have to be identified. This information can be obtained from electrogram data collected during medical procedures and from physiological literature in which population data has been analyzed and summarized. Due to limited interactions with the patient (e.g. during a device implantation procedure or an ablation procedure), currently the quality and quantity of patient-specific physiological data is sparse as there is generally not enough information to identify all the parameters in the heart model. A model with the spatio-temporal structure that is similar to the conduction patterns in the heart helps simplify the process of identifying the model parameters. A rigorous procedure for the model identification step is an important contributor to the model validation

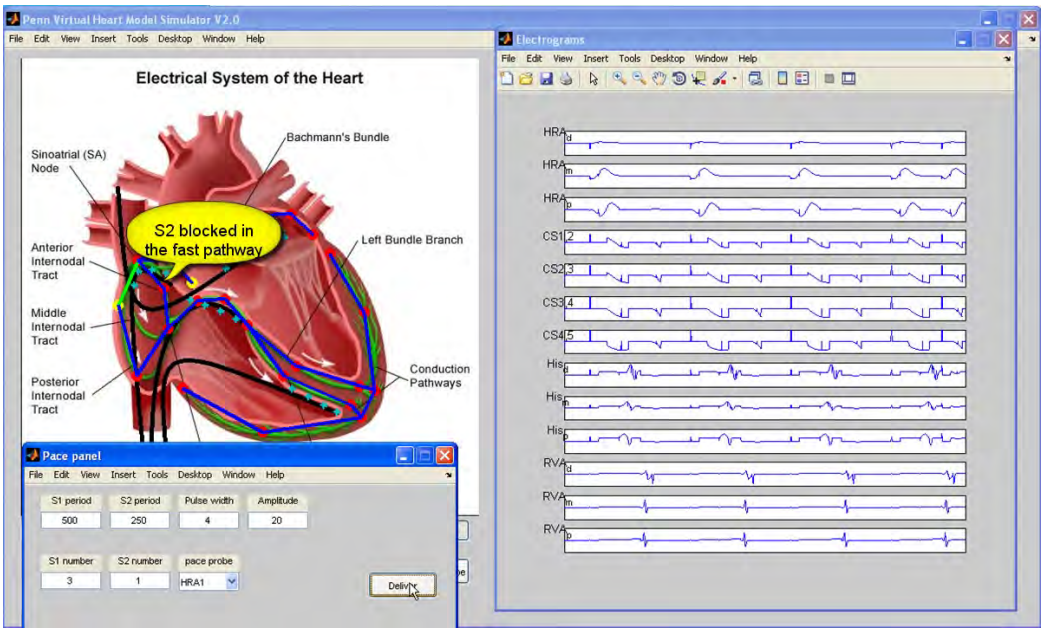


Figure 3.12: Simulation model of the heart showing the conduction pathways (left) with electrogram signals from different probe locations (right) and an interactive pacing panel (bottom left). In this case, the heart was paced four times at an interval of 500ms, followed by a pacing at a shorter (250ms) interval. This EP Testing procedure is employed to trigger conduction along alternative pathways and check for the existence of a reentry circuit.

step. In this section, we briefly discuss our model identification effort for heart models used in two closed-loop validation applications, and their corresponding challenges.

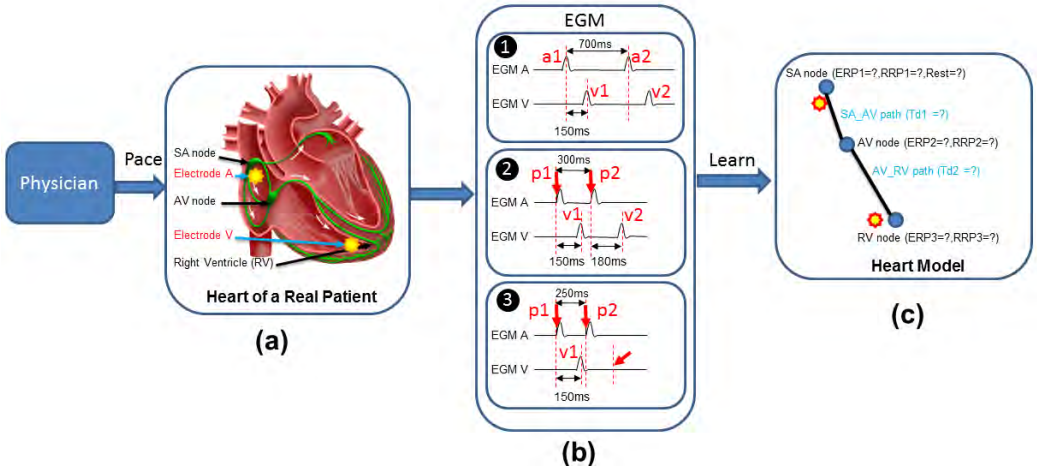


Figure 3.13: (a) The illustration of the probe locations. (b) Multiple pacing sequences with different timing outcomes. (c) The heart model with undecided parameters

3.6.1 Heart Model Identification for Closed-loop Testing

In closed-loop simulation, a deterministic heart model should be identified to represent a specific patient under a certain heart condition. The constraints for model parameters can be obtained from patient data with *Electrophysiological (EP) Testing*. During EP testing, the physician delivers electrical pacing sequences from electrodes placed inside the patient's heart to instigate responses along fast and slow conduction pathways (Fig. 3.12). The observed patterns and timing of electrical events are used to extract conduction and propagation properties of different tissue regions across the myocardium. Since the goal for any EP testing procedure is not to determine all the timing parameters for a patient, the number of parameters that can be identified from the patient data is limited.

Fig. 3.13 illustrates how timing parameters can be extracted during an EP testing procedure. Fig. 3.13(a) shows a setup with two electrodes placed in the right atrium and right ventricle of the heart respectively. EGM signals can be measured from these two electrodes (Fig. 3.13(b)). The physician delivers a series of long interval pacing sequences followed by one or more short interval pacing through the electrodes. This may trigger different responses along primary and alternate conduction pathways from the patient's heart. Fig. 3.13(c) shows a heart model structure with unknown parameter values. By analyzing the *timing* and *pattern* of the EGM signals we extract constraints on the heart model parameters. In EGM sequence 1, the interval between two intrinsic activations $a1$ and $a2$ in EGM A is 700ms, so we have:

$$ERP1 + RRP1 + Rest = 700ms$$

The interval between $a1$ and $v1$ is 150ms, so we have:

$$Td1 + Td2 = 150ms$$

In EGM sequence 2, the pacing interval from Electrode A is 300ms. By observing that the interval between $p1 - v1$ is less than the interval between $p2 - v2$, we know that $p2$ arrives during the RRP period of the AV node. So we have:

$$ERP1 + RRP1 \leq 300ms$$

In EGM sequence 3, the pacing interval is further reduced to 250ms. There is no $v2$ corresponding to $p2$, indicating $p2$ arrives during the ERP period of the AV node. So we have:

$$ERP1 \leq 250ms$$

Each experiment provides additional time constraints for model parameters. By systematically conducting experiments certain model parameters can be uniquely identified within a relatively tight range. However, even with simplified model structure like the one in the example, not all model parameters can be uniquely identified due to limited number of electrodes and limited number of experiments during a real procedure.

TABLE 2-1 Normal Conduction Intervals in Adults

Laboratory	P-A	A-H	H-V	H
Narula (2,5)	25–60	50–120	35–45	25
Damato (1,3,18,28)	24–45	60–140	30–55	10–15
Castellanos (6)	20–50	50–120	25–55	
Schuilenburg (23,24)	85–150	35–55		
Peuch (4,14)	30–55	45–100	35–55	
Bekheit (25,26)	10–50	50–125	35–45	15–25
Rosen (27)	9–45	54–130	31–55	
Author	60–125	35–55	10–25	

Figure 3.14: Timing intervals measured during clinical studies (Josephson [2008])

Heart Model Identification in Closed-loop Model Checking

In model checking, the heart models have simpler structure and fewer parameters due to non-deterministic abstraction. The placement and connectivity of nodes and paths in the heart models are developed to be consistent with EP practice. This way, each node and path automata and their timing parameters have physiological correspondence to parameters found in literature (Fig. 3.14). The range for non-deterministic parameters directly corresponds to the range for possible values of the respective physiological parameters. Therefore, model identification for model checking is much simpler and requires less EP testing data. It is important to note here that model checking of abstract models of the closed-loop system and testing of the device in the loop are complementary approaches for validating the safety and efficacy of the overall system.

3.7 Discussion

The quality and validity of closed-loop validation results depend on the quality and validity of the physiological models. It is essential for the physiological models to be complex enough to accurately interpret the behaviors of physiological conditions, and abstract enough to be identifiable from patient data. In order to evaluate the devices in closed-loop, the models should also be simple enough so that they can be used for model checking and interact with device implementations in real-time.

In this chapter, an EP heart model structure is developed for closed-loop evaluation of implantable cardiac devices. The heart model structure is capable of representing a large variety of heart conditions and its parameters can be identified from patient data.

However, the heart model structure can describe electrical activities of the heart, thus heart conditions related to other perspectives (i.e. mechanical) cannot be inferred from the model.

In the remaining dissertation, the heart model structure will be used in different model-based techniques to provide safety and efficacy evidence for implantable cardiac devices.

Chapter 4

Theme 2: Closed-loop Model Checking for Implantable Pacemaker

Model checking is a technique in which the state space of the model under investigation is automatically and exhaustively explored to identify executions or states that violate specified properties. Violations of the properties are returned by the model checkers as *counter-examples*, which can be used by designers to revise the design. In this chapter, model checking is used to evaluate an early design of a dual chamber pacemaker. More specifically, model checking is used to identify known and unknown physiological hazards induced by implantable pacemakers (e.g. when the pacemaker provides inappropriate therapy which drives the heart to an unsafe state).

The chapter is organized as follow: first the basis for timed-automata formalism is introduced. The dual chamber pacemaker specification introduced in Chapter 2 is implemented in model checker UPPAAL. The heart model structure in Chapter 3 is adjusted for closed-loop model checking of the pacemaker model. An abstraction tree of heart models is constructed to capture heart behaviors for different heart conditions and provide physiological contexts to counter-examples. Finally the abstraction tree is used to check safety and efficacy properties of the pacemaker model, as well as the effectiveness of additional algorithms developed to mitigate two known safety hazards. We demonstrated that with appropriate models of the physiological environment, closed-loop model checking is capable of finding safety and/or efficacy violations within autonomous medical devices during device development.

4.1 Related Work

Jee et. al present a safety assured development approach of real-time software using pacemaker as their case study in Jee et al. [2010b]. They formally model and verify a single chamber VVI pacemaker using UPPAAL and then implement it and check

the preservation of properties transferred from model to implementation code.

Chen et. al Chen et al. [2014] extended our verification work (Jiang et al. [2012b]). They developed a hybrid heart model which is able to simulate action potential at tissue level. The model is a more refined model than our Virtual Heart Model (Jiang et al. [2012a]), with linear dynamics on each state of the heart tissue. They also developed a probability model to simulate natural pacemaker function. They then used the combined heart model for quantitative verification of the pacemaker. However, since the pacemaker only sense the timing of the heart tissue activation, their hybrid extension for action potential does not bring much benefit but increased model complexity dramatically. As a result, they have to use bounded model checking thus sacrificed accuracy.

Tuan et. al propose an RTS formal model for pacemaker and its environment and verified it against number of safety properties and timed constraints using the PAT model checker (Tuan et al. [2010]). They have modeled the pacemaker for all 18 operating modes as described in Boston scientific, but their work lacks specification and analysis of complex behaviors of the pacemaker, such as mode-switch.

Wiggelinkhuizen uses mCRL2 and UPPAAL to formally model the pacemaker from the firmware design of Vitatron's DA+ pacemaker (Wiggelinkhuizen [2007]). Two main approaches have been used to investigate the feasibility of applying formal model checking to the design of device firmware. The main approach consists of verifying the firmware model in context of a formal heart model and a formal model of a hardware module which fails for high heart rates because of the state explosion. Another approach is to verify a part of firmware design which was feasible and was able to detect a known deadlock rather soon.

Macedo et. al have developed a concurrent and distributed real-time model for a cardiac pacemaker through a pragmatic incremental approach (D. et al. [2008]). The models are expressed using the VDM and are validated primarily by scenario-based test, where test scenarios are defined to model interesting situations such as the absence of input pulses. The models cover 8 modes of pacemaker operation.

Gomes et. al present a formal specification of pacemaker system using the Z notation in Gomes and Oliveira [2009a]. They have also tried to validate that the formal specification satisfies the informal requirements of Boston Scientific by using a theorem prover, ProofPower-Z. They have partially checked the consistency of their specification through reasoning. No validation experiment regarding safety conditions were performed yet.

Mery et. al in Mery and Singh [2009], formally model all operational modes of a single electrode pacemaker system using event-B and prove them. They use an incremental proof-based approach to refine the basic abstract model of the system and add more functional and timing properties. They use the ProB tool to validate their models in different situations such as absence of input pulses.

4.2 Model of A Dual Chamber pacemaker

During development of a dual chamber pacemaker, the specification can be translated into a model so that it can be analyzed by model checkers. The pacemaker specification discussed in Chapter 2 utilizes the timing and patterns of electrical events in the heart, which can be intuitively modeled by timed-automata (Alur and Dill [1994]) and evaluated using model checker UPPAAL Larsen et al. [1997].

4.2.1 Timed Automata

Timed automata (Alur and Dill [1994]) is an extension of a finite automaton with a finite set of real-valued clocks. It has been used for modeling and verifying systems which are triggered by events and have timing constraints between events. UPPAAL is a standard tool for modeling and verification of real-time systems, based on networks of timed automata. The graphical and text-based interface makes modeling more intuitive. Safety and efficacy requirements can be specified using Computational Tree Logic (CTL), as described in Clarke and Emerson [1982], and violations can be visualized in the simulation environment.

Syntax of Timed Automata

A timed automaton $\mathbf{G}$ is a tuple $\langle S, S_0, \Sigma, X, inv, E\rangle$, where

- S is a finite set of locations.
- $S_0 \in S$ is the set of initial locations.
- Σ is the set of events.
- X is the set of clocks.
- inv is the set of invariants for clock constraints at each location.
- E is the set of edges. Each edge is a tuple $\langle s, \sigma, \Psi, \lambda, s'\rangle$ which consists of a source location s, an event $\sigma \in \Sigma$, clock constraints Ψ, λ as a set of clocks to be reset and the target location s'.

For the clock variables X, the clock constraints $\Psi \in \Psi^X$ can be inductively defined by $\Psi := x \perp c \| \Psi_1 \wedge \Psi_2$, where $\perp \in \{\leq, =, \geq\}$, and $c \in \mathbb{N}$.

Semantics of Timed Automata

A state of a timed automaton is a pair $\langle s, v\rangle$ which contains the location $s \in S$ and the valuation v for all clocks. The set of all states is Ω. For all $\lambda \in X$, $v[\lambda := 0]$ denotes the valuation which sets all clocks $x \in \lambda$ as zero and the rest of the clocks

unchanged. For all $t \in \mathbf{R}$, $v + t$ denotes the valuation which increase all the clock value by t. There are two kinds of transitions between states. The discrete transition happens when the condition of an edge has been met. So we have:

$$\langle s, \sigma, \Psi, \lambda, s' \rangle \in E, v \models \Psi, v[\lambda := 0] \models inv(s')$$

$$\Rightarrow (s, v) \xrightarrow{\sigma} (s', v[\lambda := 0])$$

The timed transition happens when the timed automaton can stay in the same location for certain amount of time. We have:

$$\delta \in R, \forall \delta' \leq \delta, v + \delta' \models inv(s)$$

$$\Rightarrow (s, v) \xrightarrow{\delta} (s, v + \delta)$$

4.2.2 UPPAAL Model of a Dual Chamber Pacemaker

The five timing cycles introduced in Chapter 2 can be modeled as timed automata in UPPAAL (Fig. 4.1). Different components communicate with each other using broadcast channels in UPPAAL. The pacemaker model takes $Aget$ and $Vget$ events as inputs from the heart model, and outputs AP and VP as pacing signals to the heart.

4.3 Heart Models for Closed-loop Model Checking

During closed-loop model checking, the device model is verified against safety and efficacy properties under physiological conditions covered by the human physiology. It is challenging to develop physiological models for closed-loop model checking of autonomous medical devices. The following aspects need to be taken into account:

- **Model Interpretability:** How much detail should the physiological models have in order to unambiguously describe a physiological behavior? In particular, if the model checker returns an execution trace as counter-example, how much detail should the physiological model have so that the execution traces can be interpreted by medical domain experts?
- **Behavior Coverage:** What approach must we use for physiological models to cover the large variability of human physiology? How can these models cover rare physiological cases and those that are unknown to us?
- **Model Ambiguity:** Multiple physiological conditions can map to the same event trace due to the limited observability of the device. How can we eliminate only the healthy execution from the model so that it would not cause a false-positive?

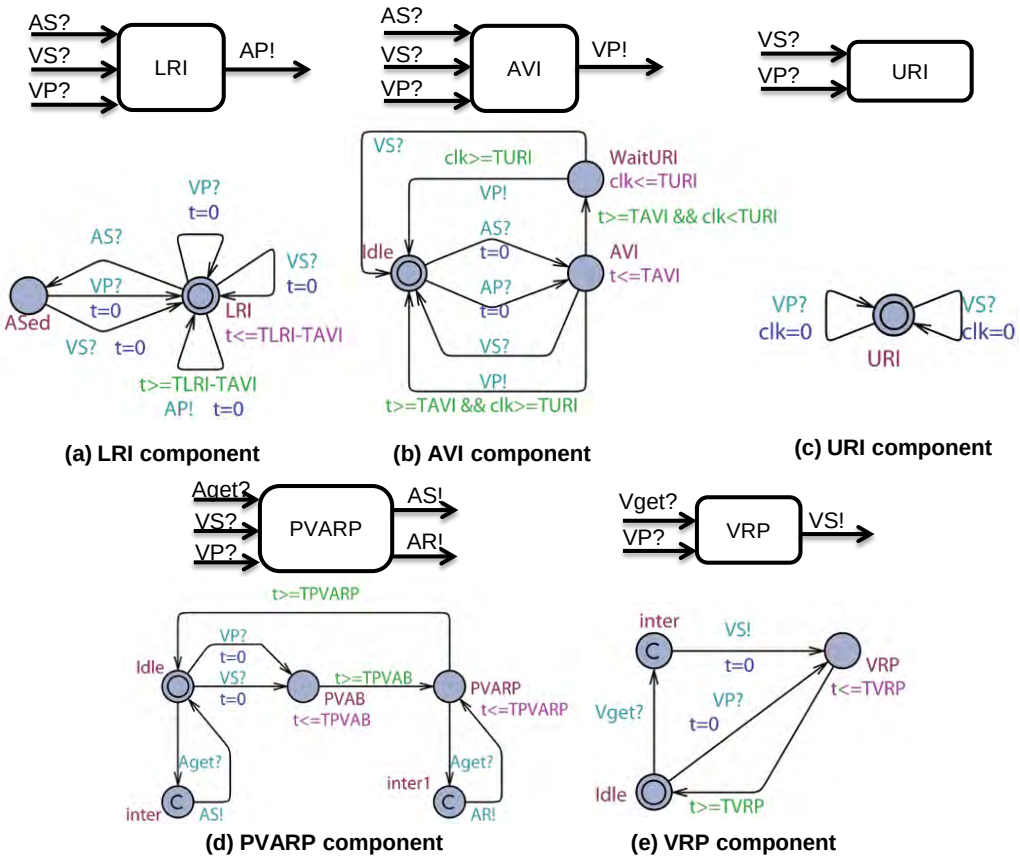


Figure 4.1: Five basic timing cycles for a dual chamber pacemaker, which include the Lower Rate Interval (LRI), Atrio-Ventricular Interval (AVI), and Upper Rate Interval (URI). Also included are the blanking intervals, Post Ventricular Atrial Refractory Period (PVARP) and Ventricular Refractory Period (VRP), to inhabit action by the pacemaker.

The heart model structure proposed in Chapter 3 can be used to model various heart conditions. The model structure provides interpretability for closed-loop interaction between the heart and the pacemaker and is capable of solving ambiguities between heart conditions that can map to the same pacemaker execution. However, physiological conditions cannot be exhaustively enumerated and model checking on all possible heart models is infeasible. In the remaining section, over-approximation is first proposed to increase behavior coverage of heart models. However, over-approximation inevitably introduces invalid behaviors into the model, which can cause false-positives. Moreover, due to the loss of details during over-approximation, the interpretability of the model decreases which prevents the model to distinguish between different heart conditions. At the end of this section, an abstraction tree framework is proposed to balance model abstraction and refinement for closed-loop model checking.

4.3.1 Covering More Behaviors With Over-approximation

Over-approximation Clarke et al. [2003] has originally been proposed to reduce model complexity during model checking. For timed-automata, timed-simulation is a form of over-approximation.

For two timed automata $T^1 = \langle S^1, S_0^1, \Sigma^1, X^1, inv^1, E^1 \rangle$ and $T^2 = \langle S^2, S_0^2, \Sigma^2, X^2, inv^2, E^2 \rangle$, a timed simulation relation is a binary relation $\mathsf{sim} \subseteq \Omega^1 \times \Omega^2$ where Ω^1 and Ω^2 are sets of states of T^1 and T^2. We say T^2 time simulates T^1 ($T^1 \preceq_t T^2$) if the following conditions holds:

- Initial states correspondence: $(\langle s_0^1, \mathbf{0} \rangle, \langle s_0^2, \mathbf{0} \rangle) \in \mathsf{sim}$
- Timed transition: For every $(\langle s_1, v_1 \rangle, \langle s_2, v_2 \rangle) \in \mathsf{sim}$, if $\langle s_1, v_1 \rangle \xrightarrow{\delta} \langle s_1, v_1 + \delta \rangle$, there exists $\langle s_2, v_2 + \delta \rangle$ such that $\langle s_2, v_2 \rangle \xrightarrow{\delta} \langle s_2, v_2 + \delta \rangle$ and $(\langle s_1, v_1 + \delta \rangle, \langle s_2, v_2 + \delta \rangle) \in \mathsf{sim}$.
- Discrete transition: For every $(\langle s_1, v_1 \rangle, \langle s_2, v_2 \rangle) \in \mathsf{sim}$, if $\langle s_1, v_1 \rangle \xrightarrow{\sigma} \langle s_1', v_1' \rangle$, there exists $\langle s_2', v_2' \rangle$ such that $\langle s_2, v_2 \rangle \xrightarrow{\sigma} \langle s_2', v_2' \rangle$ and $(\langle s_1', v_1' \rangle, \langle s_2', v_2' \rangle) \in \mathsf{sim}$.

Certain properties are preserved for timed simulation relation. For $\varphi \in ATCTL$, if $M \preceq_t M'$, we have $M' \models \varphi \Rightarrow M \models \varphi$ Yamane [2006]. However, $M' \not\models \varphi \Rightarrow M \not\models \varphi$ does not hold. Violations of $ATCTL$ yield counter-examples and the validity of which need to be checked.

In this work, the property of introducing additional behaviors is used as our advantage to cover more behaviors into a more abstract heart model. By applying physiological abstraction rules to heart models, we creates over-approximation of the heart models that not only covers all behaviors of the original heart model, but also covers additional behaviors that belong to other heart conditions.

It is known that timed simulation relation is also closed under composition Yamane [2006]. So when we have two heart models $H_1 \preceq_t H_2$ we will have $H_1 \| P \preceq_t H_2 \| P$ where P is the timed-automata model of the pacemaker. For $\varphi \in ATCTL$, we have $H_2 \| P \models \varphi \Rightarrow H_1 \| P \models \varphi$.

4.3.2 Counter-Example-Guided Abstraction Refinement

The abstract heart models obtained by over-approximation can be used for closed-loop model checking of the device model. If the closed-loop system satisfies a requirement, the device under verification satisfies the requirement under environment conditions covered by the abstract models. However, if the requirement is not satisfied, the model checker returns a counter-example. In device modeling, the counter-example is considered *spurious* if it can not be produced by the device (as shown in (Fig. 4.2(a)). However in environment modeling, even if the counter-example can not be produced by any of the initial environment models, it might still be a physiologically valid

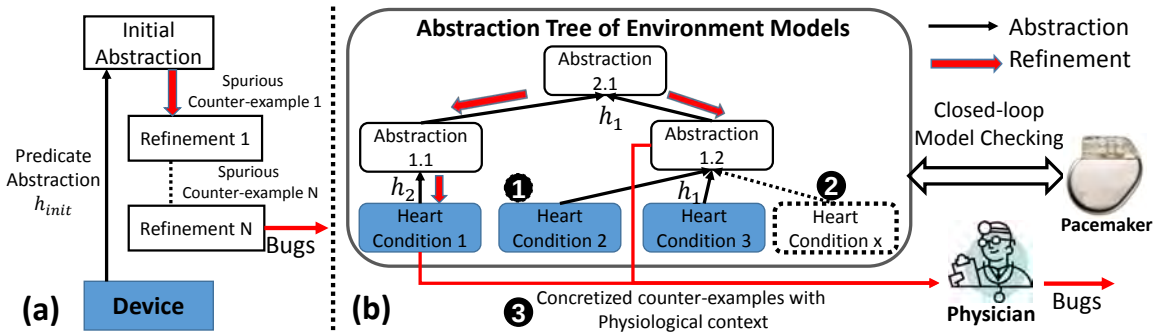


Figure 4.2: (a) Device modeling with CEGAR framework (b) Closed-loop model checking with environment abstraction tree.

behavior. Thus the validity of a counter-example cannot be determined by refining the environment model, but can ultimately only be determined by domain experts.

Counter-examples returned from abstract models can be difficult to interpret by domain experts. One abstract counter-example could be produced by multiple physiologically valid conditions, which causes ambiguity. Thus, a rigorous framework is necessary to balance the need to cover a wide range of environmental conditions and the need to provide counter-examples to the physicians within their physiological context.

Another challenge for closed-loop model checking of medical devices is the amount of domain expertise needed during: 1) physiological modeling, 2) model abstraction and refinement, and 3) checking the validity of counter-examples. Thus the framework must also allow non-domain experts to perform verification (item 2 above), and establish 'hand-off' points where the results of verification can be handed back to the experts for interpretation.

4.3.3 Abstraction Tree for Heart Model Abstraction Refinement

The ideal heart model for closed-loop model checking of an implantable pacemaker not only covers all possible inputs to the pacemaker, but also has physiological explanations to all known heart conditions. However, no **single** heart model can satisfy both requirements. Therefore, a set of heart models must be employed where the different abstraction levels of the models strike a balance between coverage and expressiveness. More importantly, the heart models should have rigorous relationships among each other to provide formal guarantees.

In this section we present the abstraction tree framework that maintains formal *Timed Simulation* relationships between heart models and enables automated closed-loop model checking of implantable pacemaker.

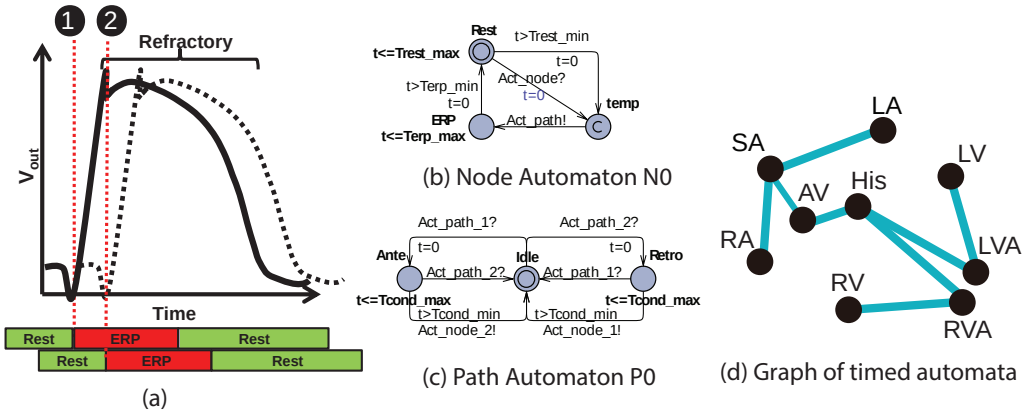


Figure 4.3: Node and Path Automata which models the timing properties of the heart tissue. A network of node and path automata models the generation and conduction of electrical activities of a heart

Initial Set of Heart Models

The heart model structure discussed in Chapter 3 is implemented in UPPAAL as shown in Fig. 4.3. Dynamic changes of the ERP periods and conduction delays are abstracted as ranges using *non-determinism* in timed-automata. This enables the heart model structure to capture behaviors of the heart models with timing variability. This heart model structure is based on clinical electrophysiology, with state variables and parameters directly corresponding to physiological parameters. Therefore, domain experts from clinical electrophysiology can construct models of different heart conditions with their domain expertise and literature.

An example set of initial heart models is shown in Fig. 4.4. The different topologies of node and path automata represent the mechanism of different heart diseases. These heart models represent the current knowledge for heart condition variability, thus the set is inherently incomplete, meaning there is no guarantee for 100% safety even if a property is satisfied in all of these models. These models are mostly used for providing physiological contexts for counter-examples returned by the model checker. Domain experts can always expand the set with knowledge of new heart conditions.

Interaction With the Pacemaker

The interactions between the heart and the pacemaker are modeled by using binary event channels. For the atrial lead, we have: $N_A.Act_path! \rightarrow$Aget!, and for ventricular lead we have $N_V.Act_path! \rightarrow$Vget!.
The pacemaker accordingly generates atrial or ventricular pacing actions AP!$\rightarrow N_A.Act\_node!$ and VP!$\rightarrow N_V.Act\_node!$.

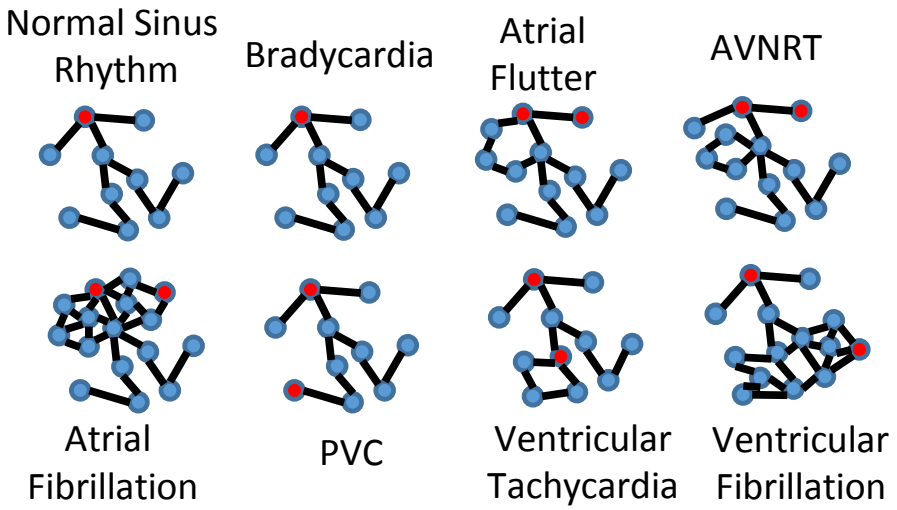


Figure 4.4: Examples of the initial set of heart models. The models are different in node and path topology and/or timing parameters.

Physiological Abstraction Rules

The initial set of heart models only represents a subset of all possible conditions. There always exists conditions that are beyond our knowledge or that are combinations of known conditions. By using *over-approximation*, heart models can be created that cover the observable behaviors of the initial set and that introduce behaviors that were not captured in the initial set. Inevitably, some of the introduced behaviors will be physiologically invalid. This problem can be alleviated by carefully designing the abstraction rules so that behaviors introduced are mostly physiologically valid. The physiologically invalid behaviors can be eliminated during a validity check in the abstraction tree.

Physiological abstraction rules are developed to cover observable behaviors of heart models. Applying one abstraction rule to heart model(s) $H_1, H_i \ldots H_n, n \geq 1$ yields an abstract heart model H' such that all observable behaviors of H_i are covered by H'. For each heart model H_i, H' is a *timed simulation* of H_i. To illustrate, a subset of abstraction rules is described intuitively. The complete set of abstraction rules and the proofs of timed simulation relationship can be found in the tech report Jiang et al. [2015].

Rule R1: Convert Reentry Circuits to Activation Nodes

Within the conduction network of the heart, there can be multiple pathways between two locations, forming conduction loops. If the timing parameters of the tissue along the loop satisfy certain properties, there can be scenarios in which a depolarization wave circling along the circuit. The circuits are referred to as *Reentry Circuits*. Since the time interval for an activation wave to circle a reentry circuit is usually less

than the intrinsic heart cycle length, the heart rate will be "'hijacked"' by the reentry circuit once the cycling is triggered, causing tachycardia. Reentry is the most common mechanism for tachycardia, which can be captured by our heart models that are used in Jiang et al. [2010a].

The effect of reentry tachycardia is that activation signals coming out of the circuit with a given cycle length equal the sum of conduction delays along the circuit. It is therefore reasonable to model a reentry circuit as a self-activation node with the self-activation range equal to the sum of conduction delays.

Applicable Condition: The rule only affects the topology of the model, and therefore can be applied without preliminaries.

Output model: The "essential structure" of a heart model is the shortest paths (in terms of conduction delay) connecting self-activation nodes and/or sensing nodes. First detect all circles in the input graph. For each circle with nodes $N_i, i \in [1 \dots n]$ and paths $P_j, j \in [1 \dots m]$, remove all "non-essential" nodes and paths, create a node automaton N_s and connect to the nearest sensing node with a path automaton P_s.

Effect on parameters: For the new node automaton N_s, the minimum of the Trest parameter is set to the minimum of the sum of the conduction delays within the reentry circuit, and the maximum is set to infinity

Effect on behaviors: The new model captures the behavior of the original model when the reentry circuit is active and inactive. Additionally, the new model captures the behaviors of other heart conditions in which the rate of the reentry circuit is lower.

Fig. 4.5 shows an example in which a circle is replaced by a self-activation node.

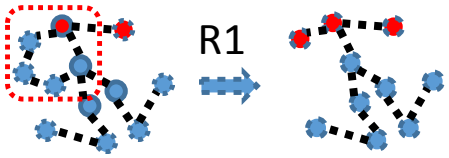


Figure 4.5: Rule R1: Remove reentry circuits from the model

Rule R2: Remove Non-essential Structures

After the circles within the topology are removed, the topology of the heart model is in the form of a tree. Since the "non-essential" structures do not affect the activation signals from and/or to the sensing nodes, all the "non-essential" structures can be removed.

Applicable Conditions: The rule can only be applied after Rule 1 has been applied.

Output model: Trimmed topology with only the essential structure remaining.

Effects on parameters: There are no effects on parameters of the node and path automata.

Effects on behaviors: Applying this rule does not affect the observable behaviors of the model.

Fig. 4.6 shows an example in which non-essential structures are removed.

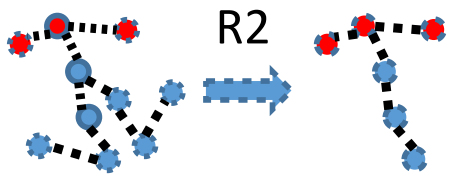


Figure 4.6: Rule R2: Remove non-essential structures

Rule R3: Removing Unnecessary Non-self-activation Nodes

The effect of non-self-activation nodes is blocking electrical events with interval shorter than its ERP period. If the self-activation nodes at both ends of a core path have self-activation interval longer than the maximum ERP period of nodes along the core path, the nodes can be removed.

For a core path from a self-activation node N_1 to another core node N_2, for any structure $P_1 - N_n - P_2$ which N_n is a non-self-activation node, if $N_n.ERP_{max} < min(N_1.Rest_{min}, N_2.Rest_{min})$, replace $P_1 - N_n - P_2$ with P_3 so that:

$$P_3.cond_{min} = P_1.cond_{min} + P_2.cond_{min}$$

$$P_3.cond_{max} = P_1.cond_{max} + P_2.cond_{max}$$

Figure 4.7: Rule R4: Merging parameter ranges

Rule R4: Merge Parameter Ranges

Timing periods of heart tissue, such as Rest and ERP, are modeled as locations in the node and path automata. The minimum and maximum time an automaton can

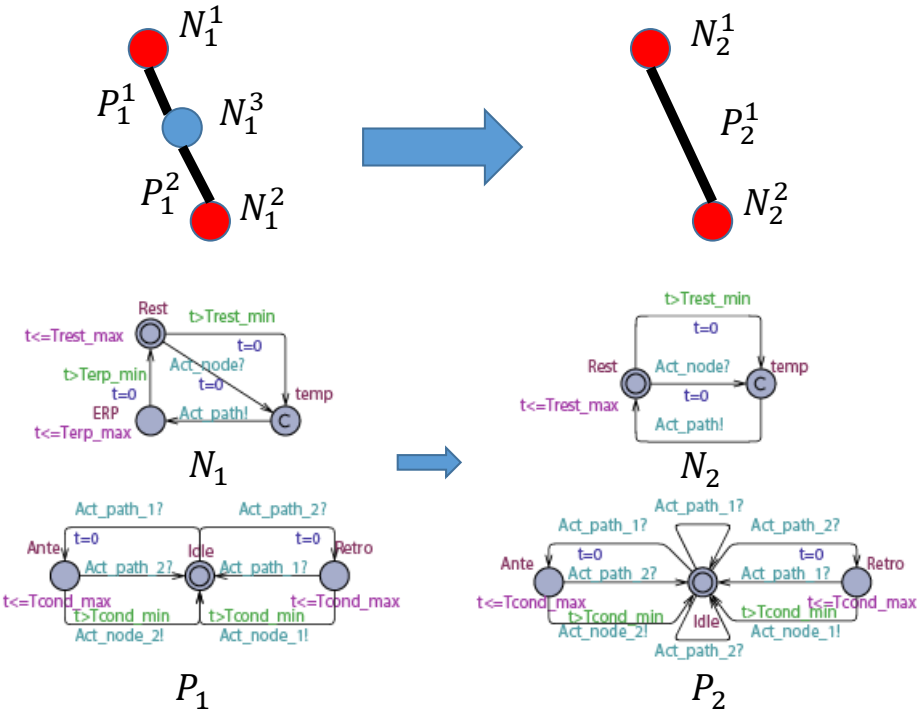


Figure 4.8: Rule R6 application example: the blocking property of N_1^3 is fulfilled by a non-deterministic conduction path P_2^1

remain in a location is governed by the parameters in the guards and invariants. By merging and expanding these periods, new behaviors are introduced where a heart model may remain longer in Rest, activate or self-activate a node faster, andsoforth.
Applicable Conditions: This rule applies to heart models with the same node and path topology but possibly with different parameters.
Output Model: The abstract model has the same topology as the original models.
Effects on parameters: The parameter ranges in the new model are a super-set of the parameter ranges in the old models.
Effects on behaviors: The abstract model captures all behaviors of the original models. In addition, heart conditions with parameters outside of the ranges of the original models are covered.

Rule R5: Merge Self-activation Nodes with Interaction Nodes

The effect of self-activation nodes on the interaction of the pacemaker is triggering sensing events within certain delay. In this rule we merge all the self-activation nodes to their neariest interaction nodes. If there exists multiple self-activation nodes merging to the same interaction node, the parameters of the new model are determined following Rule 3.

Rule R6: Replace Blocking With Non-deterministic Conduction

Consider the structure $N_1^1 P_1^1 N_1^3 P_1^2 N_1^2$ with three nodes and two paths, where N_1^3 is a passive node (i.e. not self-activating). If N_1^3 blocks an activation signal from N_1^1 to N_1^2, this is equivalent to the paths P_1^1 or P_1^2 not conducting. In this rule, the structure $P_1^1 N_1^3 P_1^2$ is replaced by a path P_2^1 whose automaton can take a self loop when it receives an activation signal, thus effectively stopping the conduction. This is shown in Fig. 4.8. Because the blocking effect of nodes is now incorporated into the paths, the node automata of self-activating nodes can be modified to the one shown in Fig. 4.8, which does not have the (now useless) ERP period.
Subgraph to which it applies. Line graphs with 3 vertices $N_1^1 P_1^1 N_1^3 P_1^2 N_1^2$, and self-activating nodes.
Applicability conditions. N_1^2 is a passive node.
Output subgraph. $N_2^1 P_2^1 N_2^2$as shown in Fig. 4.8
Effect on parameters For the new path, $P.cond_{min} = P_1.cond_{min} + P_2.cond_{min}$ and $P.cond_{max} = P_1.cond_{max} + P_2.cond_{max}$ For the new nodes, $N'.Trest_{min} = N.Terp_{min} + N.Trest_{min}$ and $N'.Trest_{max} = N.Terp_{max} + N.Trest_{max}$.

Rule R7: Replace Conduction With Self-activation

We describe Rule R7 as it illustrates both effects of an abstraction rule: structure change and modifications to the automata. The effect of a conduction path is to conduct electrical activity from a node. Since the pacemaker cannot distinguish self-activation of the node and activation triggered by path conduction, we can use self-activation to replace path conduction. If all self-activation nodes are allowed at any time by setting their minimum Rest period to 0, all the conduction paths can be removed, while preserving the original behaviors (where the Rest period was constrained to a finite interval).
Applicability conditions. This rule can only be applied after Rule 5 and Rule 6 have been applied.
Output graph. All edges are deleted: $G' = (V(G), \emptyset)$. The node automata are replaced with the one shown in Fig. 4.9.d.
Effect on parameters For every node automaton N in G', $N.Trest_{min} = 0$.

Now we use R7 as example to demonstrate the timed-simulation relationship between heart models before and after the application of R7. Consider Fig. 4.9.(a) showing an application of R7, $H_{vt}''' = N_A^1 P^1 N_V^1$ is abstracted to $H_{all} = N_A^2 N_V^2$. Here we prove that $H_{vt}''' \preceq_t H_{all}$ with observable events $\Sigma_o = \{N_A.Act\_path, N_V.act\_path\}$. The state of H_{vt}''' is represented by $(N_A^1.loc, P_1^1.loc, N_V^1.loc, N_A^1.t, P_1^1.t, N_V^1.t)$ and the state of H_{all} is represented by $(N_A^2.loc, N_V^2.loc, N_A^2.t, N_V^2.t)$. Due to space limit, only one transition from each category is presented:

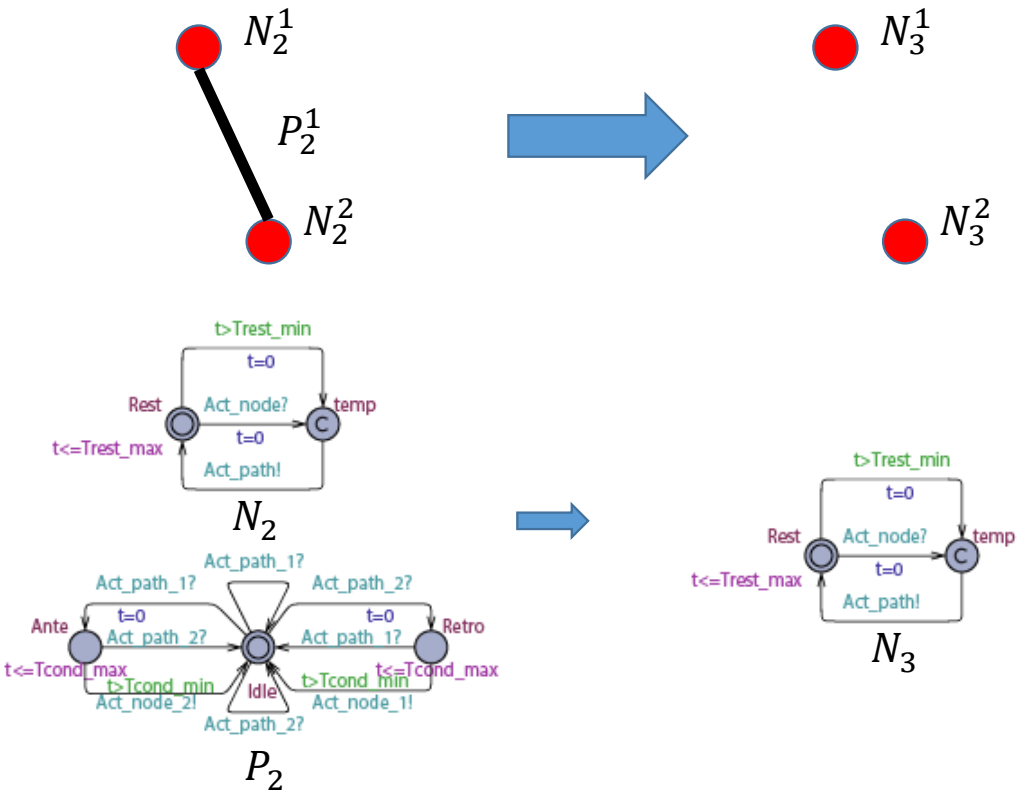


Figure 4.9: Rule R7 application example: The conduction property of P_2^1 is replaced by self activation of N_3^1 and N_3^2

Initial state: First for the initial state we have:

$$\langle (Rest, Idle, Rest, 0, 0, 0), (Rest, Rest, 0, 0) \rangle \in sim_o$$

Timed transitions: Consider a timed transition in H_{vt}'''

$$(Rest, Idle, Rest, t_1, t_2, t_3) \xrightarrow{\tau} (Rest, Idle, Rest, t_1 + \tau, t_2 + \tau, t_3 + \tau)$$

in which $(\tau \in \mathbb{R}) \wedge (t_1 + \tau \leq N_A^1.Trest\_max) \wedge (t_3 + \tau \leq N_V^1.Trest\_max)$. For a state in H_{all} such that $\langle (Rest, Idle, Rest, t_1, t_2, t_3), (Rest, Rest, t_1, t_3) \rangle \in sim_o$, there is a timed transition:

$$(Rest, Rest, t_1, t_3) \xrightarrow{\tau} (Rest, Rest, t_1 + \tau, t_3 + \tau)$$

and $\langle (Rest, Idle, Rest, t_1 + \tau, t_2 + \tau, t_3 + \tau), (Rest, Rest, t_1 + \tau, t_3 + \tau) \rangle \in sim_o$.
Discrete transitions: Consider a discrete transition in H_{vt}'''

$$(Rest, Ante, Rest, t_1, t_2, t_3) \xrightarrow[t_2 \in [P_1^1.Tcond\_min, P_1^1.Tcond\_max)]{N_V^1.Act\_path!} (Rest, Idle, Rest, t_1, t_2, 0)$$

in which $N_V^1.Act\_path! \in \Sigma_o$.
For a state in H_{all} such that $\langle (Rest, Idle, Rest, t_1, t_2, t_3), (Rest, Rest, t_1, t_3) \rangle \in sim_o$, there is a discrete transition:

$$(Rest, Rest, t_1, t_3) \xrightarrow[t_3 \in [0, N_V^2.Trest\_max)]{N_V^2.Act\_path!} (Rest, Rest, t_1, 0)$$

and $\langle ((Rest, Idle, Rest, t_1, t_2, 0)), (Rest, Rest, t_1, 0) \rangle \in sim_o$. Basically activation due to conduction is replaced by self-activation of the corresponding node automata.
Additional behaviors: The timed-simulation also allows additional behaviors into H_{all}. Consider a discrete transition in H_{all}

$$(Rest, Rest, t_1, t_3) \xrightarrow[t_3 \in [0, N_V^2.Trest\_min)]{N_V^2.Act\_path!} (Rest, Rest, t_1, 0)$$

However, for a state in H_{vt}''' such that $\langle (Rest, Idle, Rest, t_1, t_2, t_3), (Rest, Rest, t_1, t_3) \rangle \in sim_o$, when $t_3 \in [0, N_V^1.Trest\_min]$ there is no available discrete transitions. Physiologically, these implicitly included behaviors correspond to fast heart rate, premature heart events and even noise.

Abstraction Tree

By applying the abstraction rules to the initial set of heart models, an abstraction tree is created. Fig. 4.10 shows an example of an abstraction tree with the root model capturing all possible input sequences to the pacemaker. Self-activating nodes are marked as red and the Trest parameters are specified next to them. Note that this abstraction tree is not unique. With a different initial set of heart models and/or different rule application orders the abstraction tree can be very different. The abstraction tree can also be extended at any time if new heart conditions are specified. The following section demonstrates the use of this abstraction tree during the closed-loop model checking of the pacemaker design.

4.4 Efficacy Validation for Implantable Pacemaker

The most essential function for the pacemaker is to treat bradycardia by maintaining the ventricular rate above a certain threshold. We define the region where the ventricular rate is slow, as unsafe. The monitor PLRI_test is designed to measure intervals between ventricular events and is shown in Fig. 4.11(a). For property

$$\varphi_{LRI} = \text{A[] (PLRI\_test.secV imply PLRI\_test.t} \leq \text{TLRI)}$$

we have a closed-loop system with heart model H_0 :

$$H_0 \| P \| PLRI\_test \models \varphi_{LRI}$$

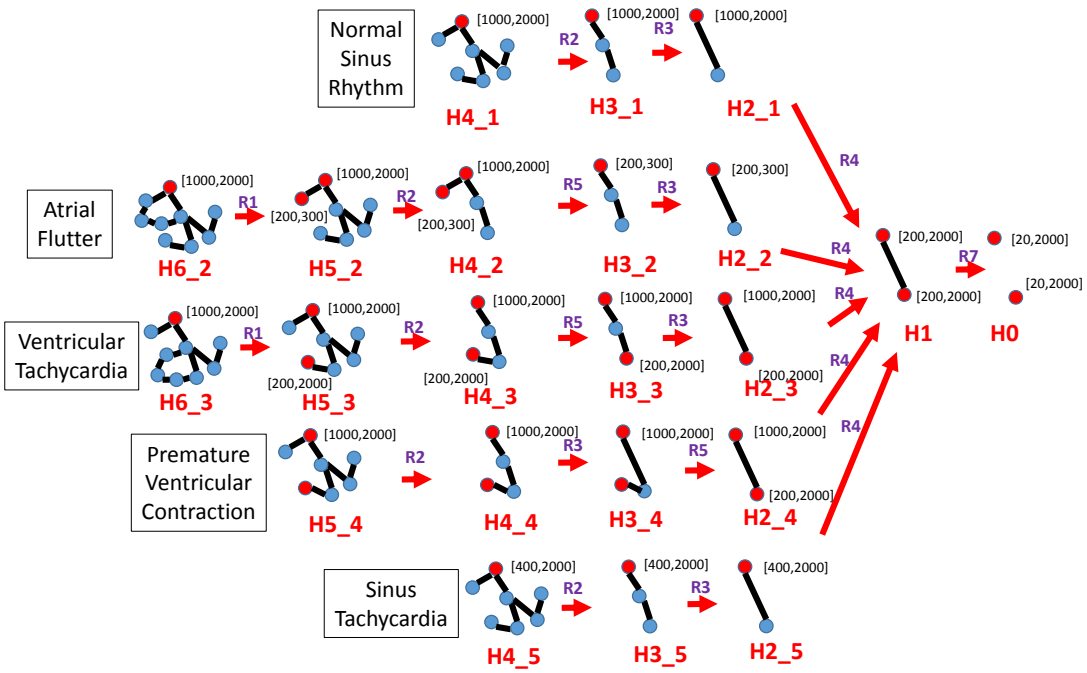


Figure 4.10: One example of abstraction tree of heart models

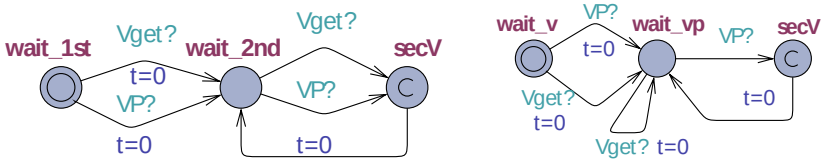
The pacemaker is not designed to treat tachycardia so it can only pace the heart to increase its rate and cannot slow it down. To mitigate the hazard that the pacemaker may increase the heart rate above physiological need, an Upper Rate Interval (URI) is specified such that the pacemaker can increase the ventricular rate up to this limit.

We require that a ventricle pace (VP) can only occur at least $TURI$ after a ventricle event (VS, VP). The monitor PURI_test is shown in Fig. 4.11(b). For the property

$$\varphi_{URI} = \text{A[] (PURI\_test.secV imply PURI\_test.t} \geq \text{TURI)}$$

we have:

$$H_0 \| P \| PURI\_test \models \varphi_{URI}$$



(a) Monitor PLRI_test (b) Monitor PURI_test

Figure 4.11: (a) Monitor for LRL: Interval between two ventricular events should be less than TLRI, (b) Monitor for URL: Interval between a ventricular event and a VP should be longer than TURI

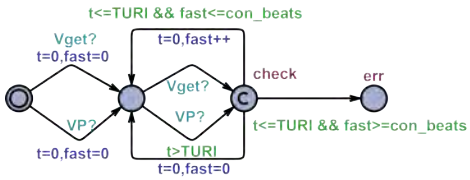


Figure 4.12: UPPAAL monitor for Property 1

As we saw in the above examples, the efficacy requirements are satisfied with the most abstract heart model. Therefore no heart model refinements are necessary and the requirements are satisfied under all possible heart conditions.

4.5 Safety Validation for Implantable Pacemaker

A dual chamber pacemaker is designed to increase the heart rate during bradycardia, but it should also not increase the heart rate inappropriately. Inappropriate increase of heart rate by the pacemaker is referred to as *Pacemaker Mediated Tachycardia (PMT)*. Previous work Jiang et al. [2014] used model checking to identify two known PMT conditions. However, in order to avoid ambiguities in the counter-examples, the properties for the two PMTs were specified very specifically, which abandoned the advantage of model checking to find unknown safety violations.

With the abstraction tree approach, the ambiguities can potentially be resolved since the abstraction tree considers all known heart conditions. Therefore the property for PMTs can be specified more generally. Here we specify a property such that
φ_{PMT}: *The interval between ventricular events (Vget, VP) should not be shorter than TURI for 30 consecutive beats*
which means that the ventricular rate should not be faster than the upper rate interval (TURI) for too long, either intrinsically or because of pacemaker interaction. Counter-example because of intrinsic fast ventricular rates can be removed from results after analysis from the abstraction tree.

A UPPAAL monitor M_{con} for the property is shown in Fig. 4.12, and the TCTL property is specified as:

$$A[]\ not\ M_{con}.err$$

The model checker UPPAAL was used to check Property φ_{PMT} on the pacemaker model using the abstraction tree of heart models. The property is violated in $H0||PM$, thus thus the abstraction tree is followed to select pair $H1$ with the pacemaker model and Property 1 is verified again. The process continues till either the leaves of the tree are reached of the property is satisfied. The result is shown in Fig. 4.13, which demonstrates 5 different scenarios that can happen when using the abstraction tree. The shaded area marks the heart models with counter-examples.

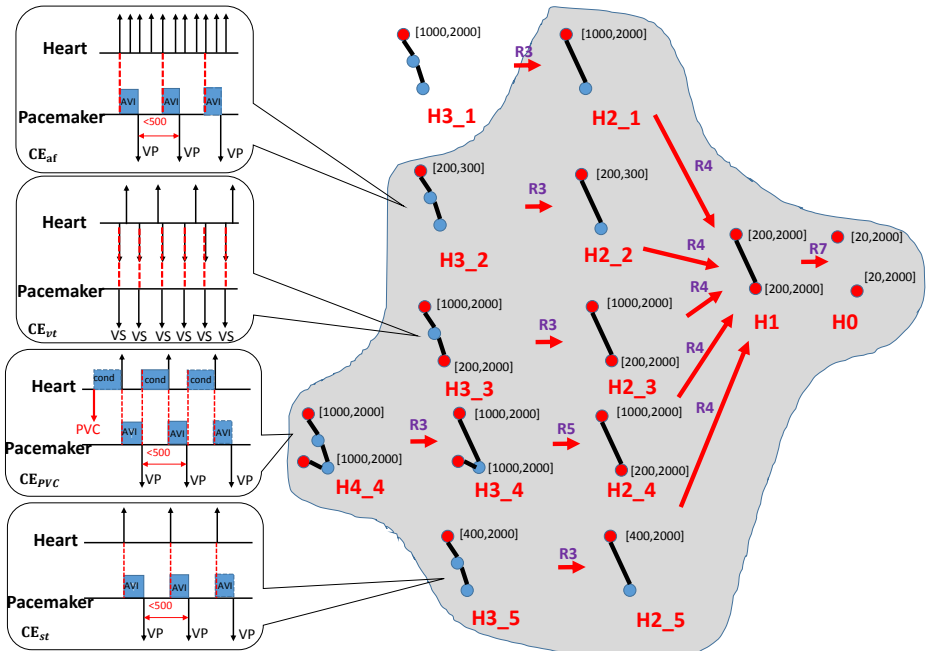


Figure 4.13: Four different physiological conditions in which Property 1 is violated. In CE_{af} the pacemaker extends a fast atrial rate to a dangerously fast ventricular rate; in CE_{vt} the ventricular rate is intrinsically fast; in CE_{st} the pacemaker appropriately maintained A-V conduction delay; in CE_{pvc} the pacemaker inappropriately increased the ventricular rate, causing Endless Loop Tachycardia

Case 1: Property 1 is violated in $H2\_1||PM$ but is satisfied in its children $H3\_1||PM$. Careful examination of the counterexample finds it to be spurious and so it is successfully eliminated by model refinement.

Case 2: CE_{af} is returned by $H3\_2||PM$ and corresponds to intrinsic atrial tachycardia with fast atrial rate, which is a sub-optimal but non-lethal condition. The AV node of the heart will block a subset of the electrical events and maintain a normal ventricular rate. However, despite the filters in the pacemaker, the pacemaker still paces the ventricle for every 3 atrial activations, which extends fast atrial rate to more dangerous fast ventricular rate. The condition is referred to as Atrial Tachycardia Response in which the ventricular rate is increased inappropriately, thus requires revision of the pacemaker design.

Case 3: CE_{vt} is returned by $H3\_3||PM$ and corresponds to intrinsic ventricular tachycardia with fast ventricular rate. The counter-example is physiologically valid but the fast ventricular rate is due to the heart itself and is beyond pacemaker functionality. Therefore this scenario demands no revision of the pacemaker design.

Case 4: CE_{st} is returned by $H3\_5||PM$ and corresponds to sinus tachycardia, for instance, when the patient is exercising. The pacemaker improved the open-loop heart condition by pacing the ventricles AVI after each atrial event, which is a correct operation of the pacemaker despite the requirement violation.

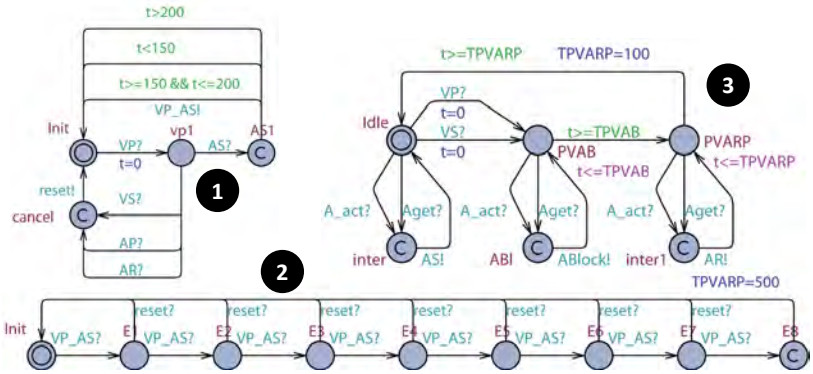


Figure 4.14: (1) The component PVAS sends VP_AS! event when a VP-AS pattern with delay between [150,200] is detected; (2) Component ELTct. After 8 VP-AS pattern, the algorithm increase TPVARP to 500ms. (3) Modified PVARP' component. TPVARP can only be set to 500 for one timing cycle.

Case 5: CE_{pvc} is returned by $H4\_4||PM$ and has a very similar input-output relationship to CE_n. However, the activations of the atrial node are triggered by retrograde conduction from ventricular paces (marker cond). The atrial activations trigger another ventricular pace after AVI, which will trigger another retrograde conduction. In this case the heart rate is inappropriately high, which corresponds to a dangerous closed-loop behavior referred to as *Endless Loop Tachycardia*.

From the above result, it can be seen that abstraction tree is able to 1) refine a heart model to eliminate spurious counter-examples as in Case 1; 2) provide (multiple) physiological explanations to a counter-example as in Case 2-5; and 3) resolve ambiguities caused by abstraction as in Case 4 and 5.

Device manufacturers have developed algorithms aiming to eliminate behaviors showed in Case 2 and Case 5. In the following section we demonstrate the use of abstraction tree to evaluate the effectiveness of these algorithms.

4.5.1 Terminating Endless Loop Tachycardia

Due to the limited information the pacemaker has about the heart, the pacemaker cannot distinguish a retrograde atrial event from an intrinsic atrial event which is triggered by the SA node. From the pacemaker's point of view, the pacemaker paces the ventricles as specified for every AS. That is why open-loop testing is unable to detect this closed-loop behavior.

Modern pacemakers are equipped with anti-ELT algorithms to identify and terminate potential ELT. One common algorithm identifies ELT by the ELT pattern and terminates ELT by increasing TPVARP time once to block the AS caused by the V-A conduction. By increasing the blocking interval after a ventricular event, the pacemaker effectively ignores the early atrial signal detected due to the PVC.

ELT termination algorithm

The ELT persists without intervention and the patient's heart is forced to beat at a fast rate approaching the Upper Rate Limit. Thus, device manufacturers require a way to identify ELT and terminate it despite the limited information the pacemaker can get. The ELT detection algorithm by Boston Scientific Boston Scientific Corporation [2007a] utilizes these three features:

- Ventricular rate at Upper Rate Limit
- VP-AS pattern
- Fixed V-A conduction delay

The pacemaker first monitors VP-AS pattern with ventricular rate at upper rate limit. Then it compares the VP-AS interval with previous intervals. ELT is confirmed if the difference between the current VP-AS interval and the first VP-AS interval is within ±32ms for 8 consecutive times. Then the pacemaker increases the PVARP period to 500ms once so that the next AS will be blocked and will not trigger a VP, terminating ELT. As the V-A conduction delays are patient-specific, the algorithm compares the VP-AS interval to a previously sensed value instead of an absolute value. Since we can not store past clock values in UPPAAL, we can not explicitly model this ELT detection algorithm. However, since the conduction delay in our heart model is within a known range, we can compare the VP-AS interval with this range. The VP-AS pattern detection module $VPAS$ for our anti-ELT algorithm is shown in Fig. 4.14 (1). It detects the VP-AS pattern with ventricular rate at the Upper Rate Limit and sends out a VP_AS event if the interval qualifies.

A counter $ELTct$ counts the number of qualified VP-AS patterns. It increases the PVARP period to 500ms if eight consecutive VP-AS patterns are detected. (Fig. 4.14 (2)) The PVARP component is also modified so that the PVARP period can only be changed once by the anti-ELT algorithm. (Fig. 4.14 (3))

Verification of the algorithm

With the new pacemaker model
$P_1 = LRI \| AVI \| URI \| PVARP' \| VRP \| ELTct \| VPAS$
we first check whether the two efficacy properties still hold when the anti-ELT algorithm is introduced. We have

$$H_0 \| P_1 \| PLRI\_test \models \varphi_{LRI}$$

$$H_0 \| P_1 \| PURI\_test \models \varphi_{URI}$$

Indicating the efficacy properties still hold after introducing the anti-ELT algorithm.

Then property φ_{PMT} is checked, we have

$$H2\_4 \| P_1 \| PELT\_det \| Pvv \not\models \varphi_{PMT}$$

which indicates the algorithm successfully eliminated all ELT executions.

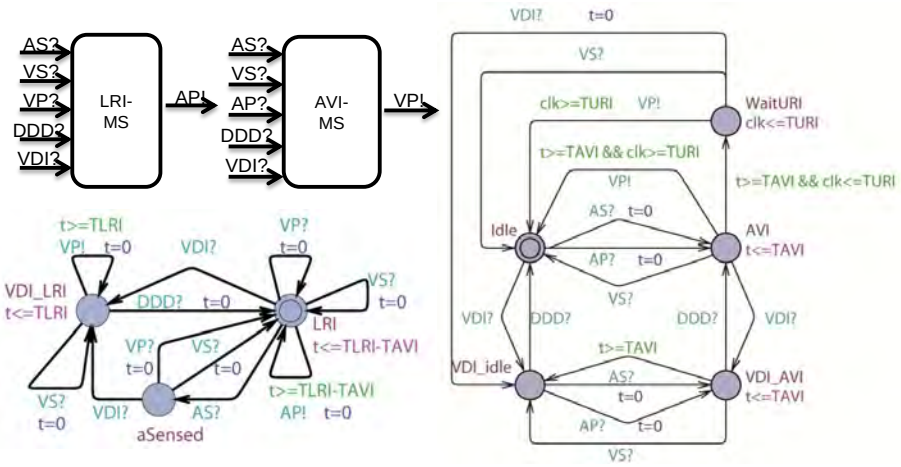


Figure 4.15: (a) After switching to VDI mode, the new LRI component LRI' maintains a minimum V-V interval; (b) After switching to VDI mode, the new AVI component AVI' keeps track of the time after each atrial events.

4.5.2 Mode Switch Operation: Atrial Tachycardia Response

Pacemaker manufacturers have designed algorithms to detect and terminate behaviors as in CE_{af}. Intuitively, the mode-switch algorithm first detects SVT. After confirmed detection, it switches the pacemaker from a dual-chamber mode to a single-chamber mode. During the single-chamber mode, the A-V synchrony function of the pacemaker is deactivated thus the ventricular rate is decoupled from the fast atrial rate. After the algorithm determines the end of SVT, it will switch the pacemaker back to the dual chamber mode. The mode-switch algorithm (also known as atrial tachycardia response) specification we use is similar to the one described in the Boston Scientific pacemakers' manual (Boston Scientific Corporation [2007b]). The algorithm first measures the interval between atrial events outside the blanking period (AS, AR). The interval is considered as *fast* if it is above a threshold (*Trigger Rate*) and *slow* otherwise. In our UPPAAL model we model it as INT (see Fig. 4.16 (1)). A counter CNT increments for *fast* events and decrements for *slow* events (see Fig. 4.16 (2)). After the counter value reaches the *Entry Count*, the algorithm will start a *Duration* (DUR) ,which is a time interval used to confirm the detection of SVT (see Fig. 4.16 (3)). In the *Duration*, the counter keeps counting. If the counter value is still positive after the *Duration*, the pacemaker will switch to the VDI mode (*Fallback mode*). In the VDI mode, the pacemaker only senses and paces the ventricle. At any time if the counter reaches zero, the *Duration* will terminate and the pacemaker is switched back to DDD mode. In our UPPAAL model of the mode-switch algorithm, we use nominal parameter values from the clinical setting. We define *trigger rate* at 170bpm (350ms), *entry count* at 8, *duration* for 8 ventricular events and *fallback mode* as VDI.

In order to model both DDD and VDI modes and the switching between them, we made modifications to the AVI and LRI components. In each component two copies for both modes are modeled, and switch between each other when switching events

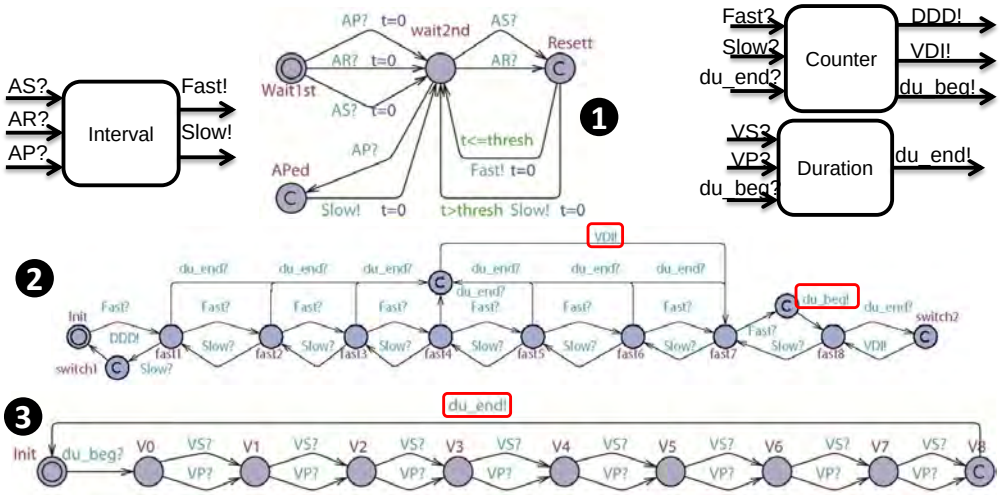


Figure 4.16: (1) Component INT: An atrial event (AS,AR) arrives before thresh after the previous atrial event is regarded as a fast event. Atrial event arrives after thresh and AP are regarded as slow event; (2) Component CNT: After 8 fast event the algorithm will start a duration by sending du_beg and will switch to VDI mode when the duration ends (du_end); (3) Component DUR :The duration length is 8 ventricular events (VS,VP)

(DDD, VDI) are received. During VDI mode, VP is delivered by the LRI component instead of the AVI component. The clock values are shared between both copies in order to preserve essential intervals even after switching. The modified AVI (AVI') and LRI (LRI')components are shown in Fig. 4.15.

Verification of the Mode-Switch Algorithm

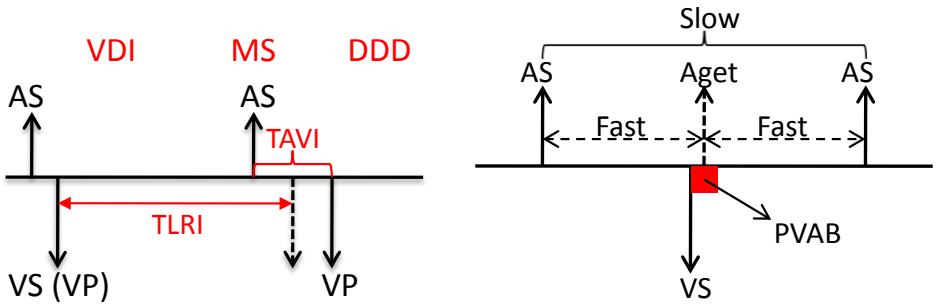
With the new pacemaker model
$P_2 = LRI' \| AVI' \| URI \| PVARP' \| VRP \| INT \| CNT \| DUR$
we first check whether the two efficacy properties still hold when the anti-ELT algorithm is introduced. We have

$$H_0 \| P_2 \| PLRI\_test \not\models \varphi_{LRI}$$

$$H_0 \| P_2 \| PURI\_test \models \varphi_{URI}$$

By analyzing the counter-example we found that when the pacemaker is switching from VDI mode to DDD mode, the responsibility to deliver VP switched from LRI component to AVI component. Since the clock reference is different (Ventricular events in LRI component and Atrial events in AVI component), the clock value for delivering the next VP is not preserved. As a result, if an atrial event which triggered the mode-switch from VDI to DDD happens within [TLRI-TAVI, TLRI) after the last ventricular event, the next ventricular pacing will be delayed by at most TAVI time, which violates the Lower Rate Limit property (Fig. 4.17(a)). Then property φ_{PMT} is checked, we have

$$H3\_2 \| P_1 \| PELT\_det \| Pvv \not\models \varphi_{PMT}$$



(a) (b)

Figure 4.17: (a) Safety Violation: VP is delayed due to the reset of timer during mode-switch, (b) Efficacy Violation: The blocking period may block some atrial events, turning two *Fast* events to one *Slow* event (Jiang et al. [2012b])

Indicating the mode-switch algorithm failed to eliminate the PMT scenario. The counter-example trace returned by UPPAAL shows that a subset of atrial events fall into the blanking period after a ventricular event (see Fig. 4.17(b)). As a result, two fast events are reduced to one slow event and mode switch may never happen. Therefore the mode-switch algorithm in our pacemaker model can not terminate all PMT behaviors as specified as certain mild PMT events are admissible.

4.6 Discussion

Model checking is not widely use in industry, in part, due to scalability issues and also because domain expertise must be a skill possessed by the verification engineer. However, with rigorous abstraction of the system and its environment, model checking can be used to identify known and even unknown mechanisms to induce hazards. In this chapter, we demonstrated the use of closed-loop model checking to identify unknown safety hazards in an implantable pacemaker. During the process we identified the need to refine the heart models to eliminate false-positives introduced during the abstraction, and demonstrated the difficulty to do so manually. The abstraction tree approach is then proposed to reduce the effort needed for both the developers and the domain experts, which makes model checking a viable approach for providing safety and effectiveness evidence. With the abstraction tree of heart models, safety and efficacy violations are identified within the pacemaker specification, which demonstrates the capability of model checking to identify issues during the early development of autonomous medical devices.

The abstraction tree approach can potentially be used in other application domains (i.e. autonomous vehicles). However, note that in our application, the most refined heart models can still be used for model checking, as the heart model structure is simple. Depending on the application, there may not exist environment models that are simple enough for model checking. And there may not exist abstraction rules that can guarantee behavior coverage.

Chapter 5

Theme 3: Verified Model to Verified Code

Model checking is performed on abstract models of the device, which is at an early stage in the development process. However, there is no guarantee that the safety and efficacy properties verified during model checking still hold in the device implementation. In this chapter, a model translation tool is developed to automatically and rigorously translate the UPPAAL device model validated during model checking into a Stateflow model, which is a step towards simulation-based testing and subsequently to code generation. This work was done in collaboration with Dr. Miroslav Pajic.

5.1 Related Work

There are only few tools that can be used for automatic implementation of timed-automata models designed in UPPAAL (e.g., Amnell et al. [2004], Hendriks [2001]). A commonly used tool is Times (Amnell et al. [2004]) that supports code generation for general platforms, extended with task support for Lego Mindstorm^{TM}platform.

The code synthesized using Times has a very simple structure, where all transitions are stored in an array. Each transition is represented with four fields: an activity flag, source and destination location ids, and a synchronization id. The transitions are evaluated in automatically generated `check_trans` function, and for the code to operate correctly the values for all clocks should be updated in a separate procedure, triggered by the system timers. Since `check_trans` performs a single evaluation for each transition (in the order specified by the array structure), to ensure that no transitions are missed the `check_trans` function has to be continuously invoked within an infinite loop, unless the code is executed on a LEGO Mindstorm RCX brick running brickOS. Thus, in the general case, the code generated with Times completely utilizes the CPU, disallowing instantiation of any other tasks. As we will show later in the paper, this is not the case for the code that is synthesized using our development framework.

Due to this array-based structure, with the code obtained using Times it is not straightforward to decouple the controller (in our case the pacemaker) and the environment (e.g., the heart). For example, to facilitate the decoupling Kim et al. [2011] propose a modification of the initial UPPAAL model by specifying the interaction between the controller and the environment using boolean shared variables. Although the solution preserves behaviors of the initial model, as pointed out by the authors, this type of manual modifications is effectively one of the most error prone aspects of the model-based development. Hence, to avoid this type of errors, it is necessary to provide modular code synthesis from verified UPPAAL models. We will later show that UPP2SF resolves this issue, since the obtained code has a modular structure (instead of maintaining an array of transitions).

Finally, the code generated from Times models does not guarantee that some aspects of the UPPAAL timed-automata semantics will be preserved (Ayoub et al. [2010]). For example, it does not ensure that the requirement for committed states is satisfied.

5.2 A Brief Overview of UPPAAL

In this section, we present an overview of the UPPAAL tool, including some of the UPPAAL extensions (Larsen et al. [1997], Behrmann et al. [2004], Bengtsson and Yi [2004]) of the timed-automata formalism from Alur [1999].

5.2.1 UPPAAL Modeling of Real-time Systems

UPPAAL supports networks of timed automata. Each automaton is a state machine, equipped with special real-valued variables called *clocks*. Clocks spontaneously increase their values with time, at the same fixed rate. Locations (i.e., states) in automata have invariants that are predicates over clocks. A location in an automaton can be active as long as its invariant is satisfied. Transitions in automata have *guards* that are predicates over clocks and variables. A transition can be taken only if its guard is true. Because clock values increase, an initially false guard can eventually become true, allowing us to model time-dependent behaviors, such as delays and timeouts. When a transition is taken, an associated *action* is executed, which can update variable values and reset clocks to integer values (possibly non-zero).

Automata in the network execute concurrently. They can communicate via shared variables, as well as via events over synchronous channels. If c is a channel, $c?$ represents receiving an event from c, while $c!$ stands for sending an event on c. In the general case, an edge from location l_1 to location l_2 can be described in a form $l_1 \xrightarrow{g,\tau,r} l_2$, if there is no synchronization over channels (τ denotes an 'empty' action), or $l_1 \xrightarrow{g,c*,r} l_2$. Here, $c*$ denotes a synchronization label over channel c (i.e., $* \in \{!, ?\}$), g represents a guard for the edge and r denotes the reset operations performed when the transition occurs.

Timed-Automata Semantics in UPPAAL

We denote with C the set of all clocks and with V the set of all data (i.e., boolean and integer) variables. A clock valuation is a function $u : C \to \mathbb{R}^+$, and we use $\mathbb{R}^C$ to denote the set of all clock valuations. A simple valuation is the function $u_0(x) = 0$, for all $x \in C$. Similarly, a data valuation is a function $v : V \to \mathbb{Z}$, while $\mathbb{Z}^V$ denotes the set of all data valuations. A valuation w, denoted by $w = (u, v)$, is a function $w : C \times V \to \mathbb{R}^+ \times \mathbb{Z}$ such that $w(x, i) = (u(x), v(i))$, for clock valuation u and data valuation v. Also, for a valuation $w = (u, v)$, $w + d$ denotes the valuation where $(w + d)(x) = (u + d)(x) = u(x) + d$ for $x \in C$, and $(w + d)(i) = v(i)$ for $i \in \mathbb{Z}$. In the rest of the paper, a clock valuation u that satisfies that for all $x \in C$, $u(x) \in \mathbb{N}_0$ will be referred to as *integer clock valuation*, while a valuation $w = (u, v)$, where u is an integer clock valuation will be referred to as an *integer valuation*.

Furthermore, let $B(C, V)$ denote the set of conjunctions over simple clock and variable conditions of the form $x \bowtie n$, $x - y \bowtie n$ or $i - j \bowtie k$, where $n \in \mathbb{N}$, $x, y \in C$, $i, j \in V$, $k \in \mathbb{Z}$ and $\bowtie \in \{\leq, \geq, =, <, >\}$.[1] Similarly, $B(C)$ denotes the set of all conjunctions over the clock variable conditions. Thus, a guard can be defined as an element of $B(C, V)$.[2] Reset operations are used to manipulate clocks and data variables. They have the form $x = n$ or $i = c_1 * j + c_2$, where $x \in C$, $i, j \in V$, $c_1, c_2 \in \mathbb{Z}$ and $n \in \mathbb{N}_0$. We use R to denote the set of all possible reset operations, and for a reset operation $r \in R$ and valuation w, $r(w)$ is the valuation obtained from w where all clocks and data variables specified in r are assigned to the values obtained from the appropriate expressions. Finally, we use K to denote the set of all channels, and $A = \{\alpha? | \alpha \in K\} \cup \{\alpha! | \alpha \in K\} \cup \{\tau\}$ to denote the set of all actions. Here, τ denotes an 'empty' action – without synchronization.

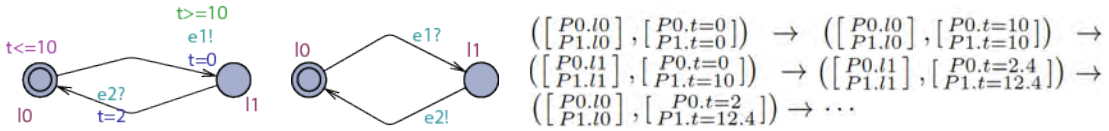
Definition 5.1. An automaton $\mathcal{A}$ is a tuple (L, l_0, A, C, V, E, I) where L denotes the set of locations in the timed automaton, l_0 is the initial location, A is a set of of actions, C a set of clocks, V is a set of data variables, and $E \subseteq L \times A \times B(C, V) \times R \times L$ denotes the set of edges, while I assigns invariants to locations (i.e., $I : L \to B(C)$ is a mapping of each location to a constraint over some clocks).

If a clock valuation u satisfies the invariants at location l, we abuse the notation and write $u \in I(l)$. Similarly, we denote with $w \in I(l)$, if $w = (u, v)$ and $u \in I(l)$. Also, if a valuation w satisfies a condition $g \in B(C, V)$ we write $w \in g$.

A network of n timed automata is obtained by composing $\mathcal{A}_i = (L_i, l_i^0, C, A, V, E_i, I_i)$, $i \in \{1, ..., n\}$. In this case, a location vector is defined as $\bar{l} = (l_1, l_2, ..., l_n)$. In addition, the invariant for location vector $\bar{l}$ is defined as $I(\bar{l}) = \wedge_i I_i(l_i)$. To denote the vector where i^{th} element of vector $\bar{l}$ (i.e., l_i) is substituted with l_i' we use the notation $\bar{l}[l_i'/l_i]$.

[1] In the latest UPPAAL versions n can also denote an expression over integer variables. Since all results from this section are valid in the latter case, we use the simplified notation where n is a constant integer.

[2] The default guard is *true*.



(a) P0 automaton (b) P1 automaton (c) A run of the model

Figure 5.1: An UPPAAL model example.

Definition 5.2. Let $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2..., \mathcal{A}_n\}$ be a network of n timed automata, and let $\bar{l}^0 = (l_1^0, l_2^0, ..., l_n^0)$ be the initial location vector. The semantics of the network is defined as a transition system $\langle \mathcal{S}, s_0, \rightarrow \rangle$, where $\mathcal{S} = (L_1 \times L_2 \times ... \times L_n) \times (\mathbb{R}^C \times \mathbb{Z}^V)$ is the set of states, $s_0 = (\bar{l}_0, w_0)$ is the initial state, where $w_0 = (u_0, v_0)$ and v_0 is any initial data valuation, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is the transition relation defined by:

1. $(\bar{l}, w) \rightarrow (\bar{l}, w + d)$ if for all d' such that $0 \leq d' \leq d$, it follows that $w + d' \in I(\bar{l})$;
2. $(\bar{l}, w) \rightarrow (\bar{l}[l_i'/l_i], w')$ if exists $l_i \xrightarrow{g,\tau,r} l_i'$ such that $w \in g$, $w' = r(w)$ and $w' \in I(\bar{l}[l_i'/l_i])$;
3. $(\bar{l}, w) \rightarrow (\bar{l}[l_j'/l_j, l_i'/l_i], w')$ if there exist $l_i \xrightarrow{g_i,c?,r_i} l_i'$ and $l_j \xrightarrow{g_j,c!,r_j} l_j'$ such that $w \in (g_i \wedge g_j)$, $w' = (r_i \cup r_j)(w)$ and $w' \in I(\bar{l}[l_j'/l_j, l_i'/l_i])$.

Since the first type of transitions is the result of time-passing, unless otherwise stated, in the rest of the paper when we use the term UPPAAL transition we will refer to either case (2) or (3) of the above definition.

To illustrate the above definition, consider the model from Fig. 5.1, where both automata have separate local clocks t. Location $P0.l0$ has invariant $t \leq 10$, while the edge $P0.l0 \rightarrow P0.l1$ has the guard condition $t \geq 10$, reset action $t = 0$, and transmission over channel $e1$. Hence, synchronization over the channel $e1$ ensures that the transitions $P0.l0 \rightarrow P0.l1$ and $P1.l0 \rightarrow P1.l1$ occur simultaneously. Finally, note that clocks do not have to be reset to zero (e.g., as on the edge $P0.l1 \rightarrow P0.l0$).

For semantics $\langle \mathcal{S}, s_0, \rightarrow \rangle$, a sequence $\mathcal{R} := (\bar{l}_0, w_0) \rightarrow (\bar{l}_1, w_1) \rightarrow ... \rightarrow (\bar{l}_i, w_i) \rightarrow ...$, is called a *run*, and we use notation $w_k^{\mathcal{R}} = w_k$, $\bar{l}_k^{\mathcal{R}} = \bar{l}_k$, for all $k \geq 0$. An example run for the model from Fig. 5.1 is shown in Fig. 5.1(c). To simplify the notation, we assume that in each run $\mathcal{R}$ no two consecutive transitions are result of time-elapsing (case (1) of Def. 5.2), since these transitions can be merged into a single transition of that type.

Additional UPPAAL Extensions of the Timed-Automata Formalism

Beside integer variables and synchronization channels, UPPAAL extends timed-automata with *committed* and *urgent* locations where time is not allowed to pass (i.e., no delay is allowed). *Committed locations* are more restrictive and they are usually used to model atomic sequences of actions. In a network of timed automata, if some automata

are in committed locations then only transitions outgoing from the committed locations are allowed. UPPAAL also introduces *broadcast channels*, where one sender can synchronize with multiple receivers (e.g., zero, one, or more than one). Furthermore, *urgent channels* can be used for synchronization, to specify that if a transition with synchronization over an urgent channel is enabled, then the transition should occur without any delay.

5.3 Extracting Runs from UPPAAL Models

To develop the UPP2SF model translation tool, we consider the problem of extracting runs for UPPAAL models. We focus on a large class of UPPAAL models without clock conditions of the form $x > E$, where x is a clock and E an expression. The restriction, while not limiting in modeling of real control system, guarantees that all invariants and guards are expressed as intersections of left semi-closed (LSC) intervals. Thus, we refer to this class as *Class LSC*, and in this work we consider only this type of models.

To obtain a run of an UPPAAL model it is sufficient to simulate the model only at integer time points (Pajic et al. [2012a]), which allows for the use of discrete-time based tools for model simulation. Since the execution of UPPAAL models is non-deterministic, a transition in UPPAAL can occur at any point at which it is enabled. However, at each time instance it may not be straightforward to determine whether a currently enabled transition will still be enabled at the next integer-time point. Therefore, to simplify the translation procedure we consider runs of UPPAAL models that are obtained using the maximal progress assumption (MPA).

Definition 5.3. For semantics $\langle \mathcal{S}, s_0, \rightarrow \rangle$, a run $\mathcal{R} := (\bar{l}_0, w_0) \rightarrow ... \rightarrow (\bar{l}_i, w_i) \rightarrow ...$, satisfies the maximal progress assumption (MPA) if for all $k \geq 0$ such that

1. $\bar{l}_{k+1} = \bar{l}_k$ and $w_{k+1} = w_k + d$ for some $d > 0$, and
2. $(\bar{l}_{k+1}, w_{k+1}) \rightarrow (\bar{l}_{k+2}, w_{k+2})$ satisfies either case (2) or (3) of Def. 5.2,

there does not exist a d', $0 \leq d' < d$, for which there exist a transition $(\bar{l}_k, w_k + d') \rightarrow (\bar{l}_k[l'_j/l_j], w')$ or a transition $(\bar{l}_k, w_k + d') \rightarrow (\bar{l}_k[l'_p/l_p, l'_q/l_q], w')$ for the given semantics.

A run that satisfies the MPA will be referred to as an *MPA run*. Note that from the definition, if some transitions in an MPA run are enabled, one of them should occur.

Theorem 5.1. *Consider an UPPAAL model from the Class LSC. For every MPA run $\mathcal{R}$ of the model and for all $k \geq 0$, the clock valuation $u_k^{\mathcal{R}}$ satisfies that for each clock x, $u_k^{\mathcal{R}}(x) \in \mathbb{N}_0$ (i.e., all transitions of $\mathcal{R}$ occur at integer time points).*

Proof. Lets assume that the theorem does not holds – i.e., there exists an MPA run $\mathcal{R}$ for which there exists $k \geq 0$ and a clock x such that $u_k^{\mathcal{R}}(x) \notin \mathbb{N}_0$. By k_0 we denote the first (i.e., lowest) such k. Since all clocks are initialized to zero, $k_0 > 0$ and $w_{k_0-1} = (u_{k_0-1}, v_{k_0-1})$ is an integer valuation. Thus, consider the part $(\bar{l}_{k_0-1}, w_{k_0-1}) \rightarrow (\bar{l}_{k_0}, w_{k_0}) \rightarrow (\bar{l}_{k_0+1}, w_{k_0+1})$ of the run $\mathcal{R}$. From Def. 5.2 one of the following cases is valid:

1. $(\bar{l}_{k_0-1}, w_{k_0-1}) \rightarrow (\bar{l}_{k_0}, w_{k_0})$ satisfies case (1) of Def. 5.2 (i.e., time passing). Then from the definition of a *run*, the transition $(\bar{l}_{k_0}, w_{k_0}) \rightarrow (\bar{l}_{k_0+1}, w_{k_0+1})$ is described by either case (2) or (3) from Def. 5.2. In the former case $\bar{l}_{k_0+1} = \bar{l}_{k_0}[l_i'/l_i]$, meaning that there exists $l_i \xrightarrow{g,\tau,r} l_i'$ such that $w_{k_0} = (u_{k_0}, v_{k_0}) \in g$, $w_{k_0+1} = r(w_{k_0})$, and $w_{k_0+1} \in I(\bar{l}_{k_0+1})$. Consider the valuation $w'_{k_0} = \lfloor w_{k_0} \rfloor = (\lfloor u_{k_0} \rfloor, v_{k_0})$. Since $w_{k_0} \in g$ then w'_{k_0} also satisfies all variable conditions from g. For each clock x, if $u_{k_0}(x)$ satisfies the clock guard conditions of the form $x \bowtie n$, where $n \in \mathbb{N}_0$ and $\bowtie \in \{\leq, \geq, =, <\}$, then $u_{k_0}(x)$ belongs to an intersection of left-closed intervals with integer boundaries. Thus, $\lfloor u_{k_0}(x) \rfloor$ belongs to the same intersection of the intervals, meaning that $u'_{k_0} = \lfloor u_{k_0} \rfloor$ satisfies this type of clock constraints from g. Furthermore, if for some clocks x and y, valuation u_{k_0} satisfies constraints of the form $x - y \bowtie n$, it follows that $u_{k_0}(x) - u_{k_0}(y) \bowtie n$. From $u_{k_0} = u_{k_0-1} + d$ (where d is the elapsed time), we have: $u'_{k_0}(x) - u'_{k_0}(y) = \lfloor u_{k_0}(x) \rfloor - \lfloor u_{k_0}(y) \rfloor = u_{k_0-1}(x) + \lfloor d \rfloor - (u_{k_0-1}(y) + \lfloor d \rfloor) = u_{k_0-1}(x) - u_{k_0-1}(y) = u_{k_0-1}(x) + d - u_{k_0-1}(y) - d = u_{k_0}(x) - u_{k_0}(y)$. Thus, $u'_{k_0}(x) - u'_{k_0}(y) \bowtie n$ is true, and all clock constraints of this form are also satisfied, implying that $w'_{k_0} \in g$. Similarly, for $w'_{k_0+1} = r(w'_{k_0}) = (u'_{k_0+1}, v'_{k_0+1})$, from $w'_{k_0} = \lfloor w_{k_0} \rfloor$ and $w_{k_0+1} = r(w_{k_0})$ it follows that $v'_{k_0+1} = v_{k_0+1}$ and $u'_{k_0+1} = \lfloor u_{k_0+1} \rfloor$ (all reset clocks are equal, since clocks are reset to integer values). Therefore, as for the guard, it can be shown that $w_{k_0+1} \in I(\bar{l}_{k_0+1})$ implies that $w'_{k_0+1} \in I(\bar{l}_{k_0+1})$, which proves that the transition $(\bar{l}_{k_0}, w'_{k_0}) \rightarrow (\bar{l}_{k_0+1}, w'_{k_0+1})$ was also enabled. Since $w'_{k_0}(x) < w_{k_0}(x)$ it follows that in this case $\mathcal{R}$ is not an MPA run, which violates our initial assumption.

   A similar proof can be used in the latter case when $(\bar{l}_{k_0}, w_{k_0}) \rightarrow (\bar{l}_{k_0+1}, w_{k_0})$ satisfies case (3) of Def. 5.2, by showing the existence of a transition enabled for $\lfloor u_{k_0} \rfloor$. Since $\lfloor u_k(x) \rfloor < u_k(x)$, the transition $(\bar{l}_{k_0}, w_{k_0}) \rightarrow (\bar{l}_{k_0+1}, w_{k_0+1})$ cannot be in an MPA run.

2. $(\bar{l}_{k_0-1}, w_{k_0-1}) \rightarrow (\bar{l}_{k_0}, w_{k_0})$ satisfies either case (2) or (3) of Def. 5.2. Then, there exists a transition with reset r such that $w_{k_0} = r(w_{k_0-1})$, or a synchronized transition with resets r_i, r_j such that $w_{k_0} = (r_i \cup r_j)(w_{k_0-1})$. However, since w_{k_0-1} is an integer valuation, in both cases $u_{k_0}(x) \in \mathbb{N}$ because no clock can be reset to a non-integer value. This conflicts our initial assumption, and thus concludes the proof.

□

The theorem presents the basis for the translation procedure. It allows us to obtain an MPA run by evaluating transitions from active locations in each automaton only at integer time points. Note that each automaton may be evaluated more than once, as more than one transition could occur within a single automaton at any integer time.

Theorem 5.1 can be easily extended for UPPAAL models with committed and urgent locations, and urgent and broadcast channels. For example, urgent locations can be modeled by adding an extra clock x_u that is reset to zero on all incoming edges to urgent locations, and adding condition $x_u \leq 0$ to invariants in all urgent locations. Yet, this does not affect the proof of Theorem 5.1, and thus the theorem is still valid. In addition, semantics for UPPAAL models that employ urgent channels is similar to the semantics from Def. 5.2, with an additional condition in case (1) of the definition. In this case $(\bar{l}, w) \rightarrow (\bar{l}, w + d)$, if for all $d', 0 \leq d' < d$ it holds that $w + d' \in I(\bar{l})$, and for any urgent channel c there does not exist $l_i \xrightarrow{g_i, c?, r_i} l_i'$ and $l_j \xrightarrow{g_j, c!, r_j} l_j'$ such that $w + d' \in (g_i \wedge g_j)$ and $(r_i \cup r_j)(w + d') \in I(\bar{l}[l_j'/l_j, l_i'/l_i])$. Similarly, broadcast channels semantics does not require that exactly one transition with receiving channel occurs simultaneously with a transition that contains a transmission over the channel – with broadcast channels none, one or more than one 'receiving' transitions could occur. Thus, Theorem 5.1 is also satisfied even if urgent and broadcast channels are used.

5.4 Brief Overview of Stateflow

A Stateflow chart (i.e., model) employs a concept of finite state machines extended with additional features, including support for different data types and events that trigger actions in a part or the whole chart. Here, we present a small subset of the Stateflow features used in the translation procedure. Detailed descriptions of other features can be found in Inc. [2016], Scaife et al. [2004], Hamon and Rushby [2007], Hamon [2005].

A state in a Stateflow chart can be active or inactive, and the activity dynamically changes based on events and conditions. States can be defined hierarchically – i.e., a state can contain other states (referred to as *substates*). A decomposition of a chart (or a state) determines if its states (substates) are *exclusive* or *parallel* states. Within a chart (or a state), no two exclusive states can be active at the same time, while any number of parallel states can be simultaneously activate (but executed sequentially).

Unlike in UPPAAL, transitions between states in Stateflow are taken as soon as enabled. They are described in the form (where each part of the description is optional)

$$Event[condition]\{condition\_actions\}/\{transition\_actions\} \tag{5.1}$$

Event identifies the event that enables the transition (which is enabled by default if *Event* is not stated), if the *condition* (if specified, by default it is true) is valid. The

condition is described using basic logical operations on conditions over chart variables and Stateflow operators. Actions in *condition_actions* and *transition_actions* include event broadcasting and operations on data variables. The Stateflow semantics specifies that when a transition from a state s_i to state s_j occurs, then *condition_action* are executed first, before the state s_i becomes inactive. This is followed by the execution of *transition_actions*, and finally activation of the state s_j (i.e., during the execution of *transition_actions* none of the states is active).

A Stateflow chart runs on a single thread and it is executed only when an event occurs. All actions that occur during an execution triggered by an event are atomic to that event. After all activities that take place based on the event are finished, the execution returns to its prior activity (i.e., activity before receiving the event). All parallel states within a chart (and similarly, all parallel substates in a state) are assigned with a unique execution order. Furthermore, all outgoing transitions from a state have different execution indices. Thus, the execution of a Stateflow chart is fully deterministic – Stateflow semantics specifies that active states are scheduled, and state transitions are evaluated in the execution order (starting from the lowest execution index).[3]

Notion of Time in Stateflow

Stateflow temporal logic can be used to control execution of a discrete-time chart in terms of time. It defines time periods using absolute-time operators based on the simulation time, or event-based operators that use the number of event occurrences. Absolute-time logic defines operators $after$, $before$ as

$$after(n, \text{sec}) = \begin{cases} 0, \text{ if } t < n \\ 1, \text{ if } t \geq n \end{cases}, \qquad before(n, \text{sec}) = not(after(n, \text{sec})) \tag{5.2}$$

where t denotes the time that has elapsed since the activation of the associated state (i.e., from the last transition to the state - including self-transitions). The value for time t can be obtained using the operator $temporalCount(\text{sec})$. Similarly, event-based temporal logic operators are used for event counting – e.g., $after(n, clk)$ returns 1 if the event clk has occurred more than $(n-1)$ times after the state has been activated.

5.5 UPP2SF: Model Translation Procedure

In this section, we present an overview of the UPP2SF translation procedure. We also describe the translation rules for UPPAAL models with urgent and broadcast channels, urgent and committed locations, and local clocks, as these functionalities are used for the pacemaker modeling. For the full UPP2SF description refer to Pajic et al. [2012c].

[3]The user can specify the execution index for each transition and state – default values are assigned by the order of instantiation. Parallel states (or transitions from a state) must have different execution indices.

5.5.1 Overview of UPP2SF

Consider an UPPAAL model with automata $P_1, ..., P_n$. The UPP2SF translation procedure would produce a two-level Stateflow chart as in Fig. 5.2, with parallel states $P_1, ..., P_n$ (referred to as the *parent* states) derived from the automata, parallel states $Gc\_x_1, ..., Gc\_x_m$ (referred to as *clock states*) that model all global clocks $x_1, ..., x_m$ from the UPPAAL model,[4] and the state *Eng* that is used as the chart's control execution engine. In addition, the chart has predefined global data variables (and constants) with appropriate variable ranges and initial values obtained from the UPPAAL model. Since all automata in UPPAAL are simultaneously active, the obtained Stateflow chart is a collection of parallel states with unique execution orders. Also, in every UPPAAL automaton exactly one location is active at a time. Thus, each of the parent states is a collection of exclusive states, extracted from locations in the UPPAAL automaton.

To ensure that the extracted chart is simulated at integer time points, input trigger event *clk* is added to the chart and a signal generator block is added to the parent Simulink model. We call a *clk execution* the execution of the chart from the moment the chart is triggered by a *clk* event, until processing of the event has been finished. Since our goal is to derive a Stateflow chart whose execution is one of the MPA runs of the initial UPPAAL model, it is possible that more than one transition within the model (and even within a single automaton) occur at any time point. Therefore, the chart can (re)activate itself by transmitting local (within the scope of the chart) events from the additional parallel state *Eng*, which is executed last of all chart's parallel states. Processing of the events triggered during a *clk execution* is considered a part of the *clk execution*. Since event processing is atomic in Stateflow, no time elapses (in Simulink) during a *clk execution* regardless how many additional event broadcasts have occurred. With this approach, a single activation of the chart triggers all transitions enabled at that integer time point, effectively extracting an MPA execution trace of the model.

Finally, any UPPAAL edge from location l_{k_i} to l_{k_j} in any automaton P_k, which does not use global clocks and synchronization over binary channels, is mapped into a Stateflow transition $P_k.l_{k_i} \rightarrow P_k.l_{k_j}$ between the corresponding substates in the parent state P_k. In the rest of the section we provide a description of the edge translation procedure.

Remark: In the general case, the edge $l_{k_i} \xrightarrow{g,\alpha,r} l_{k_j}$ (where $\alpha \in \{\tau, c!, c?\}$ and c is a binary or broadcast channel) is mapped into a more complex structure in Stateflow between the substates $P_k.l_{k_i}$ and $P_k.l_{k_j}$. If the edge uses global clocks then a junction J_{ij} and transition $P_k.l_{k_i} \rightarrow P_k.J_{ij}$ are introduced to update global clock values used in the edge's guard and invariants. Also, if α does not use a binary channel, a single transition is introduced from J_{ij} to $P_k.l_{k_j}$. However, if α uses a binary channel, to preserve the semantics three edges and a junction are added between J_{ij} and $P_k.l_{k_j}$.

[4]Note that if no global clocks are used in the UPPAAL model, the obtained Stateflow chart would not contain parallel global clocks states Gc_x_j.

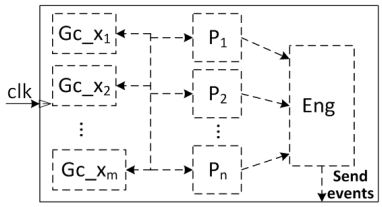


Figure 5.2: Structure of Stateflow charts derived by UPP2SF. Parent states $P_1, ..., P_n$ are derived from automata, while the *clock* states $Gc\_x_1, ..., Gc\_x_m$ model all global clocks $x_1, ..., x_m$ from the UPPAAL model. The state Eng is used to control execution of the chart.

5.5.2 Mapping UPPAAL Edges Without Synchronization

Consider an UPPAAL edge $l_{k_i} \xrightarrow{g,\tau,r} l_{k_j}$ in automaton P_k. The guard g can be split into a conjunction of data and clock conditions, and thus during the translation UPP2SF introduces a Stateflow transition $P_k.l_{k_i} \rightarrow P_k.l_{k_j}$ of the form:

$$[\underbrace{G_C(I(l_{k_i}) \wedge g) \wedge G_V(g) \wedge G_C(r, I(l_{k_j}))}_{G_{C,V}(l_{k_i}, l_{k_j}, g, r)}]/\{\underbrace{R_V(r); R_C(r);}_{R_{C,V}(r)} R_S(r);\} \tag{5.3}$$

where:

1. $G_C(h)$ ($G_V(h)$) translates the clock (data) conditions from UPPAAL condition h into an equivalent Stateflow condition,
2. $R_C(r)$ ($R_V(r)$) maps clock (data) resets in r to an equivalent Stateflow assignment,
3. $G_C(r, I(l_{k_j}))$ maps the condition that the clock valuation after the reset r satisfies the invariant at the 'new' location l_{k_j},
4. $R_S(r)$ controls execution of the chart.

Data resets ($R_V(r)$) and guard conditions ($G_V(g)$) are directly mapped into the identical Stateflow expressions. Mapping local clocks' resets and guards is described below.

Mapping Clock Conditions and Resets

In UPPAAL, each clock condition $h \in B(C)$ is specified as $h = h_1 \wedge h_2 \wedge ... \wedge h_M$, where h_i's are basic clock conditions. Therefore, $G_C(h) = G_C(h_1) \wedge G_C(h_2) \wedge ... \wedge G_C(h_M)$, and it is only necessary to provide a set of rules for the translation of basic clock conditions of the form $x \bowtie n$ or $x - y \bowtie n$, where $x, y \in C$ and $n \in \mathbb{Z}$ (or an expression over integer variables and constants).

To specify conditions over clocks, UPP2SF employs event based Stateflow temporal logic operators that (only) count the number of *clk* event occurrences. When the

UPPAAL condition ($x \in C, n \in \mathbf{N}_0$)	Stateflow condition – x replaced by $u^{\mathcal{S}}(x)$
$x \leq n$	$(temporalCount(clk) \leq n - n_x)$
$x < n$	$before(n - n_x, clk)$
$x = n$	$(temporalCount(clk) == n - n_x)$
$x \geq n$	$after(n - n_x, clk)$
$x - y \bowtie n$	$n_x - n_y \bowtie n$

Table 5.1: Mapping UPPAAL conditions over clocks into Stateflow

temporal logic operators are used in a chart with the two-level hierarchy shown in Fig. 5.2, Stateflow associates a unique counter with each parallel (i.e., parent) state. It is important to highlight here that Stateflow semantics specifies that when the appropriate event activates the chart (i.e., when *clk* triggers the chart) all these counters are incremented at the **beginning** of the chart's execution - i.e., even before the first parallel state begins its execution. Consequently, when each of the parallel states is executed, the counter value is equal to the number of the event's appearances from the activation of its currently active substate. For each parent state P_k this values is $temporalCount(clk)$, and thus we denote this value by tC^{P_k}. In addition, we define $tC^x = tC^{P_k}$ if x is a local clock defined in the automaton P_k.

Unlike in UPPAAL where clocks might not be reset to zero during a transition, for a parallel state in the Stateflow chart the aforementioned counter is always reset when a transition occurs (when the associated substate is activated). Thus, while mapping edges from the automaton P_k, we explicitly model **each local** (from P_k) **clock** x by introducing the accounting variable n_x that maintains the clock value from the moment of the last state activation. This is done using $R_C(r)$ from (5.3), which is specified as

$$[R_C(r)](x) = \begin{cases} n_x = n_x + temporalCount(clk), & \text{if } x \notin r \\ n_x = r(x), & \text{if } x \in r \end{cases} \tag{5.4}$$

Our goal is that at integer time points UPPAAL valuation u of the clock x (i.e., $u(x)$) is equal to the value $u^{\mathcal{S}}(x)$ defined as $u^{\mathcal{S}}(x) = n_x + tC^x$ (we will show this in the next section). Note that a single counter value is used for all local clocks defined within the same automaton (i.e., $tC^x = tC^y$ if x, y are local clocks defined in automaton P_k).

The transformation of the basic clock conditions presented in Table 5.1 employs event-based temporal logic operators while taking into account the values of the accounting variables for all used clocks. In the mapping each clock x is replaced with the value $u^{\mathcal{S}}(x)$. In addition, we used a relationship between Stateflow temporal logic operators from (5.2) to simplify the notation. For example, the condition $x < n$ for a local clock x is mapped into $n_x + temporalCount(clk) < n$, which is equivalent to $before(n - n_x, clk)$ (since $before(n - n_x, clk) = 1 \Leftrightarrow temporalCount(clk) < n - n_x$).

Finally, as specified in Def. 5.2, the requirement that the new clock valuation satisfies the invariant at the (new) location l_{k_j} is equivalent to the condition that both the non-reset and the reset clock values satisfy the clock invariants at location

l_{k_j}. Hence, if $I(l_{k_j}) = h_1 \wedge ... \wedge h_k$, then $G_C(r, I(l_{k_j})) = G_C(r, h_1)) \wedge ... \wedge G_C(r, h_k))$ where

$$G_C(r, x \bowtie n) = \begin{cases} r(x) \bowtie n, & \text{if } x \in r \\ u^{\mathcal{S}}(x) \bowtie n & \text{if } x \notin r \end{cases}, G_C(r, x-y \bowtie n) = \begin{cases} r(x) - r(y) \bowtie n, & \text{if } x, y \in r \\ r(x) - u^{\mathcal{S}}(y) \bowtie n, & \text{if } x \in r, y \notin r \\ u^{\mathcal{S}}(x) - r(y) \bowtie n, & \text{if } x \notin r, y \in r \\ u^{\mathcal{S}}(x) - u^{\mathcal{S}}(y) \bowtie n & \text{if } x, y \notin r \end{cases} \tag{5.5}$$

The expression for $G_C(r, I(l_{k_j}))$ can be significantly simplified. If $r(x)$ resets the clock x to a constant (which is the prevailing case in UPPAAL), conditions from (5.5) can be evaluated during the translation and can be replaced with fixed terms ($false$ or $true$).

5.5.3 Obtaining an MPA Execution of the Chart

The execution semantics of Stateflow ensures that in each of the parent states transitions from the active state will be evaluated at least once during a *clk* execution. However, to obtain an MPA run of the model, after a transition occurs it is necessary that in each parent state transitions from the active state are reevaluated. We guarantee this by reactivating the chart if at least one transition has occurred. Thus, in the chart, UPP2SF introduces the parallel state *Eng* (Fig. 5.2), which is executed last among the parent (and clock) states. Furthermore, additional chart event *tt* and flag *act* are defined, and as a part of each transition, by adding $act = 1$; to $R_S(r)$ from (5.3), *act* is set to 1. Finally, *Eng* contains a single substate and it broadcasts the event *tt* to the chart if *act* has been set to 1, using the lowest priority self-transition of the form

$$[act == 1]\{act = 0; send(tt)\} \tag{5.6}$$

5.5.4 Translating Broadcast Channels

Events in Stateflow are a good semantic match for broadcast channels in UPPAAL. Therefore, for each broadcast channel c, UPP2SF defines a Stateflow event c assigned with a unique positive integer $ID(c)$. To translate edge $l_i \xrightarrow{g,c!,r} l_j$ from automaton P_k, UPP2SF uses a centralized approach where the *Eng* state broadcasts events and controls execution of the chart by using additional variable *sent* that can have the following values (here, $ExO(P_k) > 0$ is the execution order of the parent state P_k)

$$sent = \begin{cases} ID(c), & \text{event } c \text{ is scheduled for broadcast} \\ 0, & \text{no event scheduled for broadcast} \\ -ExO(P_k), & \text{an event is broadcast, only receiving edges are enabled} \end{cases} \tag{5.7}$$

Note that $ID(c) > 0$, and $ExO(P_i) \neq ExO(P_j)$ if $P_i \neq P_j$.

Table 5.2 shows the mapping of UPPAAL edges into Stateflow. Action $c!$ is mapped into $sent = ID(C)$ assignment, thus disabling all 'non-receiving' transitions

UPPAAL edge	Stateflow transition
$l_i \xrightarrow{g,\tau,r} l_j$	$[(sent == 0) \wedge G_{C,V}(l_i, g, r, l_j)]/\{R_{C,V}(r); R_S(r);\}$
$l_i \xrightarrow{g_j,c!,r_j} l_j$	$[(sent == 0) \wedge G_{C,V}(l_i, g, r, l_j)]/\{R_{C,V}(r); sent = ID(c);\}$
$l_i \xrightarrow{g_i,c?,r_i} l_j$	$c[(sent \sim= -ExO(P_k)) \wedge G_{C,V}(l_i, g, r, l_j)]/\{R_{C,V}(r);\}$

Table 5.2: Mapping UPPAAL edges from automaton P_k into Stateflow transitions

due to their condition ($sent == 0$). Similarly, condition ($sent \sim= -ExO(P_k)$) disables all 'receiving' transitions in the parent state P_k, ensuring that the parent state does not synchronize with itself. Finally, the *Eng* state is used to broadcast events by adding for each event c the following self-transition in the state:

$$[(sent == ID(c))]\{sent = -ExO(P_k); send(c);\} \tag{5.8}$$

In addition, to reset *sent* and to ensure that all previously disabled transitions are reevaluated by reactivating the chart after the event is processed (i.e., after all parent states are re-executed), UPP2SF adds the following self-transition in the state *Eng*

$$[(sent < 0)]\{sent = 0; send(tt);\} \tag{5.9}$$

Transitions (5.8), (5.9) have precedence (i.e., lower execution order) over the transition (5.6).

Remark: In general, more than one UPPAAL automaton could transmit over a shared broadcast channel. In this case, *Eng* state would not always be able to determine the parent state P_k that has initiated the event broadcast. Thus, variable $ExOP$ would have to be defined along with additional reset action $ExOP = ExO(P_k)$ in transitions with $sent = ID(c)$; (from Table 5.2). Also, transition (5.8) would take the form $[(sent == ID(c))]\{sent = -ExOP; send(c);\}$. However, since this case does not occur in most UPPAAL models, due to the space limitation we present the simpler formulation.

5.5.5 Translating Urgent and Committed States

UPP2SF also preserves semantics of urgent and committed states, and urgent channels. By extracting MPA runs of the UPPAAL model we ensure that no time passes in the states from which there exists an enabled transitions. Thus, as a byproduct, semantics of urgent channels and locations are preserved. On the other hand, if some automata in UPPAAL are in committed locations, then only transitions outgoing from one of the committed locations are allowed. Thus, to deal with committed locations we introduce a new 'control' variable *comm* that always contains the number of active committed states. For all transitions incoming to a committed state expression $comm = comm + 1;$ is added to the reset operations (i.e., $R_S(r)$ from (5.3)). Similarly, for all outgoing transitions from a committed state $comm = comm - 1;$ is added to the reset. To disable transitions from non-committed states when there

exists an active committed state, guard condition ($comm == 0$) is added to all 'non-receiving' transitions outgoing from a non-committed state. Note that setting *act* to 1 (as specified in (5.6), for all transitions in parent states) reactivates the chart to ensure that all transitions are reevaluated, including the ones that have been disabled due to ($comm == 0$) condition.

5.5.6 Stateflow Chart Optimization

Stateflow charts obtained using the described set of rules can usually be significantly simplified. For example, clock guards and invariants specify fixed left-closed intervals if in conditions from Table 5.1 n denotes a constant. These intervals can be expressed in Stateflow with maximum two terms from Table 5.1 (e.g., invariant $t \leq n$ and guard $t \geq n$ can be combined into a single Stateflow condition $temporalCount(clk) == n - n_t$). In addition, it is possible to remove updates to an accounting variable n_x from transitions incoming to a state, if on all paths from the state there exist resets of the clock x before the clock is used in a transition guard or invariant. Similarly, due to (5.9) there is no need to reactivate the chart with the $act = 1$ reset on transitions to a committed/urgent state that are conditioned with event receiving. The same holds if outgoing transitions from a new state are disabled (which is a common case) at the time of activation, and no shared variable has been updated on the incoming transition.

5.6 Correctness of the Translation Procedure

In this section, we show that the set of rules specified in the previous section preserves the UPPAAL semantics. Specifically, we show that the execution of the obtained Stateflow chart presents one of MPA runs of the initial UPPAAL model. However, since the Stateflow semantics is informally defined, formally proving correctness of the translation procedure is not possible. For a subset of Simulink features, there exist some attempts to derive formal semantics (e.g., Hamon and Rushby [2007], Hamon [2005]), which have been validated by testing on many examples. We follow a similar approach in this work. We start by formulating basic assumptions on the semantics of the Stateflow charts obtained by UPP2SF – i.e., with the structure shown in Fig. 5.2 and which utilize only a small subset of Stateflow functionalities.

Consider a deadlock-free UPPAAL model with automata $P_1, ..., P_n$, where each automaton has at least one location, and the extracted two-level Stateflow chart as in Fig. 5.2, with parent states $P_1, ..., P_n$ and the parallel state Eng. Since none of the chart's parallel states has transitions[5] the following proposition holds.

Proposition 5.2. *All parallel states in the chart will always be active.*

[5]Parallel states in Stateflow do not typically use transitions (Inc. [2016], Hamon and Rushby [2007]).

The translation rules specify that all transitions in parent states do not have *condition_action* (from (5.1)), and no event broadcasting is specified in *transition_actions*. Thus, when a transition $P_k.l \rightarrow P_k.l'$ occurs in a parent state P_k, substate l' will be directly activated (i.e., activity will be directly passed from l to l').[6] In addition, transitions in the *Eng* state do not have *transition_actions* and they broadcasts events as part of *condition_actions*.[7] That means that the state $Eng.l0$ will never be deactivated.

Proposition 5.3. *Each parallel state in the chart always has an active substate.*

In the general case, event broadcasting to the whole chart introduces recursive behavior (we will also see this later, in Section 5.9.1, during the analysis of the pacemaker code – Listing 5, Fig. 5.3). These recursions are in general very difficult to control and analyze, and Hamon and Rushby [2007] restricted the use of events to the definition of sequencing behaviors. In UPP2SF-derived charts, only *Eng* can broadcast events, on self-transitions from the (only) substate $Eng.l0$. Since *Eng* state is executed last within each chart activation, and within its execution only a single transition may occur, local event broadcasts can **only occur at the end of chart activations**. Therefore, although event broadcasts from the self-transitions in *Eng* state introduce recursive behavior to the chart, we can consider these recursive runs as series of sequential chart activations with different active events – i.e., we can disregard the recursive behavior.

Consequently, we can describe the state of the chart as a $\theta^{\mathcal{S}} = (\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}})$, where $\bar{l}^{\mathcal{S}}$ is the vector of size n containing active substates for each of the n parent states.[8] In addition, $w^{\mathcal{S}} = (u^{\mathcal{S}}, v^{\mathcal{S}})$ where $u^{\mathcal{S}}$ is defined in (5.5.2) and $v^{\mathcal{S}}$ denotes the Stateflow variables mapped from UPPAAL variables – i.e., $v^{\mathcal{S}}(i)$ is the value of the Stateflow variable i. Finally, $f^{\mathcal{S}} = (act, sent, comm, AE, k)$ denotes the values of all control flags introduced by UPP2SF. AE ($AE \in \{clk, tt, \phi, SCE\} \cup \{c | c \in K\}$) is the currently active event being processed by the chart,[9] while k, $k \in \{0, 1, ..., n, n+1\}$, is the state index of the currently executed parallel state.[10] We denote by $\bar{l}^{\mathcal{S}}[l_{k_j}/l_{k_i}]$ a vector

[6] Broadcasting events in transition actions would result in deactivation of the substate l and event broadcasting before the substate l' is activated. Thus, during the event's processing (including processing events sent during the event's processing) the parent state P_k would not have active substates and would be effectively removed from the execution. On the other hand, broadcasting events in *condition_actions* would usually result in an infinite behavior (since in most cases the *condition* would still be satisfied (Inc. [2016]).)

[7] Note that in this case infinite cycle behavior does not occur, since the data values enabling a transition guard are changed before broadcasts – this disables the transition in the next activation.

[8] Note that since *Eng* state has a single substate that is always active, we do not specify it in the vector $\bar{l}^{\mathcal{S}}$.

[9] Here, ϕ denotes the case when no event is active - when the chart is sleeping, between consecutive *clk* executions, while *SCE* is the Simulink Call Event, an intrinsic way for Simulink to activate a Stateflow chart.

[10] In general, f should also contain a *transition_index* denoting the transition (from the active substate of the parallel state k) which is being evaluated. However, to simplify our notation we have omitted this term.

where the active substate of the parent state P_k in vector $\bar{l}^{\mathcal{S}}$ has changed from l_{k_i} to l_{k_j}, and use a similar notation for $w^{\mathcal{S}}$ and $f^{\mathcal{S}}$ updates – e.g., $f^{\mathcal{S}}[k = k + 1]$ denotes $f^{\mathcal{S}}$ where only k is increased by 1. For UPPAAL models we use the notation from Sec. 5.2, and for $w = (u, v)$ we write $w = w^{\mathcal{S}}$ (and $u = u^{\mathcal{S}}, v = v^{\mathcal{S}}$) if $\forall x \in C, i \in V$, $u(x) = u^{\mathcal{S}}(x)$ and $v(i) = v^{\mathcal{S}}(i)$.

We can now formalize the behavior (i.e., semantics) of extracted Stateflow charts. From translation rules, the initial vector of active substates $\bar{l}_0^{\mathcal{S}}$ is equal to the initial location vector in UPPAAL (i.e., $\bar{l}_0^{\mathcal{S}} = \bar{l}_0$), and $w_0^{\mathcal{S}} = w_0$ (w_0 is the initial UPPAAL valuation), $f_0^{\mathcal{S}} = (0, 0, 0, SCE, 1)$ ($AE = SCE$, as charts are executed during initialization).

Definition 5.4. A *transition relation* for a UPP2SF-derived chart is defined as:

1. $(\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}) \to (\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}[u^{\mathcal{S}} = u^{\mathcal{S}} + 1], f^{\mathcal{S}}[k = k + 1])$ if $k = 0$ and $AE = clk$,
2. $(\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}) \to (\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}[AE = clk])$ if $k = 0$ and $AE = \phi$,
3. $(\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}) \to (\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}[k = k + 1])$ if $k = 0$ and $AE \notin \{clk, \phi\}$,
4. $(\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}) \to (\bar{l}_1^{\mathcal{S}}, w_1^{\mathcal{S}}, f_1^{\mathcal{S}})$ if $k \in \{1, ..., n\}$, where

$$(\bar{l}_1^{\mathcal{S}}, w_1^{\mathcal{S}}, f_1^{\mathcal{S}}) = \begin{cases} (\bar{l}^{\mathcal{S}}[l_{k_{i_0}}/l_k], R_{C,V}^{i_0}[w^{\mathcal{S}}], R_S^{i_0}[f^{\mathcal{S}}][k = k + 1]), \\ \exists i, (G_s^i \wedge G_{C,V}^i) = True, \text{and} \\ (e^i \text{ not specified or } AE = e^i) \\ i_0 = \min i \\ (\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}[k = k + 1]), & \text{otherwise} \end{cases}$$

   and all transitions outgoing from the active substate l_k (in P_k) are represented as $e^i[G_s^i \wedge G_{C,V}^i]/\{R_{C,V}^i; R_S^i;\}$ (i is the transition index, and for example, $R_{C,V}^{i_0}[w^{\mathcal{S}}]$ denotes the value of $w^{\mathcal{S}}$ after reset operations specified in $R_{C,V}^{i_0}$ are performed on $w^{\mathcal{S}}$);

5. $(\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f^{\mathcal{S}}) \to (\bar{l}^{\mathcal{S}}, w^{\mathcal{S}}, f_1^{\mathcal{S}})$ if $k = n + 1$, where

$$f_1^{\mathcal{S}} = \begin{cases} R_S^{i_0}[f^{\mathcal{S}}][k = 0, AE = e^i], & \exists i, G_s^i = True, \text{ where } i_0 = \min i \\ f^{\mathcal{S}}[k = 0, AE = \phi]), & \text{otherwise} \end{cases}$$

   and all self-transitions in *Eng* state are described as $[G_s^i]\{R_S^i; send(e^i);\}$.

We define the chart's execution trace $\mathcal{R}^{\mathcal{S}}$ as a sequence of consecutive chart states $\theta_0^{\mathcal{S}} \to \theta_1^{\mathcal{S}} \ldots \to \theta_t^{\mathcal{S}} \to \ldots$. We also denote by $\theta_{t_0}^{\mathcal{S}} \xrightarrow{*} \theta_{t_i}^{\mathcal{S}}$ (referred to as an *SF-transition*) a sequence $\theta_{t_0}^{\mathcal{S}} \to \theta_{t_1}^{\mathcal{S}} \ldots \to \theta_{t_i}^{\mathcal{S}}$ of the execution such that one of the following holds:

- the sequence does not contain any transitions mapped from UPPAAL edges, and $k = 1, AE = clk$ in $\theta_{t_i}^{\mathcal{S}}$; in this case we denote the sequence by $\theta_{t_0}^{\mathcal{S}} \xrightarrow{*(+d)} \theta_{t_i}^{\mathcal{S}}$, since (as none of the transitions in parent states has occurred during the sequence) for some $d \in \mathbb{N}$ (when $i > 0$), for all $x, y \in C, u_{t_i}^{\mathcal{S}}(x) - u_{t_0}^{\mathcal{S}}(x) = u_{t_i}^{\mathcal{S}}(y) - u_{t_0}^{\mathcal{S}}(y) = d$.

- $k \neq 0$ or $AE \neq clk$ in all $\theta^{\mathcal{S}}_{t_j}, j \leq i-1$; also, $\theta^{\mathcal{S}}_{t_{i-1}} \rightarrow \theta^{\mathcal{S}}_{t_i}$ is a transition mapped from some UPPAAL edge $l_f \xrightarrow{g,\tau,r} l_j$, and it is the only transition in the sequence mapped from any UPPAAL edge; in this case we also use the notation $\theta^{\mathcal{S}}_{t_0} \xrightarrow{*\tau} \theta^{\mathcal{S}}_{t_i}$,
- $k \neq 0$ or $AE \neq clk$ in all $\theta^{\mathcal{S}}_{t_j}, j \leq i-1$, and the sequence contains exactly one transition mapped from some UPPAAL edge $l_f \xrightarrow{g,b!,r} l_j$, which is the first transition in the sequence that is mapped from any UPPAAL edge; followed by a number of transitions (i.e., 0, 1 or more) mapped from UPPAAL edges of the form $l_p \xrightarrow{g,b?,r} l_q$ (i.e., receiving over channel b), where $\theta^{\mathcal{S}}_{t_{i-1}} \rightarrow \theta^{\mathcal{S}}_{t_i}$ is the last of them; in addition, the next transition in the execution trace which is mapped from an UPPAAL edge is not mapped from an edge $l_n \xrightarrow{g,b?,r} l_m$; here, we also use the notation $\theta^{\mathcal{S}}_{t_0} \xrightarrow{*b!?} \theta^{\mathcal{S}}_{t_i}$.

From the above definition, the following lemma follows directly.

Lemma 5.4. *If $\theta^{\mathcal{S}}_{t_0} \xrightarrow{*\tau} \theta^{\mathcal{S}}_{t_i}$ then for all $n \in \{t_0, ..., t_i - 1\}$, $\bar{l}^{\mathcal{S}}_n = \bar{l}^{\mathcal{S}}_{t_0}$ and $w^{\mathcal{S}}_n = w^{\mathcal{S}}_{t_0}$.*

Proposition 5.5. *Assume that $\theta^{\mathcal{S}}_{t_0} \xrightarrow{*\tau} \theta^{\mathcal{S}}_{t_i}$, and $\bar{l}^{\mathcal{S}}_{t_0} = \bar{l}$ and $w^{\mathcal{S}}_{t_0} = w$. If $\theta^{\mathcal{S}}_{t_{i-1}} \rightarrow \theta^{\mathcal{S}}_{t_i}$ is a transition mapped from UPPAAL edge $l_{k_f} \xrightarrow{g,\tau,r} l_{k_j}$ in automaton P_k (k is the state index from f), then $(\bar{l}, w) \rightarrow (\bar{l}[l_{k_j}/l_{k_f}], r(w))$ in UPPAAL, and $r(w) = w^{\mathcal{S}}_{t_i}$, $\bar{l}[l_{k_j}/l_{k_f}] = \bar{l}^{\mathcal{S}}_{t_i}$.*

Proof. From Lemma 5.4, $\bar{l}^{\mathcal{S}}_{t_i-1} = \bar{l}^{\mathcal{S}}_{t_0} = \bar{l}$ and $w^{\mathcal{S}}_{t_i-1} = w^{\mathcal{S}}_{t_0} = w$. Since $\theta^{\mathcal{S}}_{t_{i-1}} \rightarrow \theta^{\mathcal{S}}_{t_i}$ is a transition mapped from $l_{k_f} \xrightarrow{g,\tau,r} l_{k_j}$, it follows that $\bar{l}[l_{k_j}/l_{k_f}] = \bar{l}^{\mathcal{S}}_{t_i}$, and from (5.3) $G_C(I(l_{k_f}) \wedge g) \wedge G_V(g) \wedge G_C(r, I(l_{k_j}))$ is satisfied. $G_V(g)$ presents the identical conditions over data variables as in the UPPAAL guard g, and since $v^{\mathcal{S}}_{t_i-1}$ satisfies $G_V(g)$, then v satisfies data conditions in g. Similarly, since $u^{\mathcal{S}}_{t_i-1}$ satisfies $G_C(I(l_{k_f}) \wedge g)$ from Table 5.1, then from $u^{\mathcal{S}}_{t_i-1} = u$ we have that u satisfies the guard and invariant at l_{k_f} – i.e., $u \in g$ and $u \in I(l_{k_f})$. Finally, $u^{\mathcal{S}}_{t_i-1}$ satisfies $G_C(r, I(l_{k_j}))$ defined in (5.5), and using the same reasoning we have that $r(u) \in I(I(l_{k_j}))$, implying $w \in g$ and $r(w) \in I(I(l_{k_j}))$.

In addition, $w^{\mathcal{S}}_{t_i} = (u^{\mathcal{S}}_{t_i}, v^{\mathcal{S}}_{t_i})$, where $v^{\mathcal{S}}_{t_i} = R_V(r)[v^{\mathcal{S}}_{t_i-1}]$ and $u^{\mathcal{S}}_{t_i} = R_C(r)[u^{\mathcal{S}}_{t_i-1}]$. Since $R_V(r)$ specifies the identical data expressions as r, $v^{\mathcal{S}}_{t_i} = r(v^{\mathcal{S}}_{t_i-1}) = r(v)$. On the other hand, when a transition occurs $temporalCount(clk)$ is reset. Thus, for all $x \in C$, $u^{\mathcal{S}}_{t_i}(x) = n_x$, and if $x \in r$ from (5.4) $u^{\mathcal{S}}_{t_i}(x) = r(x)$; otherwise $n_x = u^{\mathcal{S}}_{t_i-1}(x)$, meaning that $u^{\mathcal{S}}_{t_i}(x) = u^{\mathcal{S}}_{t_i-1}(x)$. Consequently, $u^{\mathcal{S}}_{t_i} = r(u)$, and $w^{\mathcal{S}}_{t_i} = r(w)$, which concludes the proof. □

The following results can be proven using similar approaches as in the above proof.

Proposition 5.6. *Assume that $\theta^{\mathcal{S}}_{t_0} \xrightarrow{*b!?} \theta^{\mathcal{S}}_{t_i}$, for a broadcast channel b, and $\bar{l}^{\mathcal{S}}_{t_0} = \bar{l}$ and $w^{\mathcal{S}}_{t_0} = w$. If the sequence contains transitions mapped from UPPAAL edges $l_j \xrightarrow{g_j,b!,r_j} l'_j$ in automaton P_j, and $l_{j_i} \xrightarrow{g_{j_i},b?,r_{j_i}} l'_{j_i}$ in automata P_{j_i} ($i = 0, ..., m$), then*

$(\bar{l}, w) \rightarrow (\bar{l}[l'_j/l_j, l'_{j_0}/l_{j_0}, ..., l'_{j_m}/l_{j_m}], (r_j \cup_{i=0}^{m} r_{j_i})(w))$ *in UPPAAL. Furthermore,* $(r_j \cup_{i=0}^{m} r_{j_i})(w) = w_{t_i}^{\mathcal{S}}$ *and* $\bar{l}[l'_j/l_j, l'_{j_0}/l_{j_0}, ..., l'_{j_m}/l_{j_m}] = \bar{l}_{t_i}^{\mathcal{S}}$.

sketch. We first show that after the transition mapped from the edge $l_j \xrightarrow{g_j, b!, r_j} l'_j$ occurs it is not possible in the obtained chart to have a 'non-receiving' transition, or a 'receiving' transition conditioned with an event c, where $c \neq b$. We prove then that $j \neq j_i$ for $i = 0, ..., m$, and $j_p \neq j_q$ for $p, q = 0, ..., m$ and $p \neq q$ (i.e., an automaton cannot synchronize with itself, or synchronize twice with another automaton for a single broadcast). After showing these properties, using a similar approach as for Prop. 5.5 we show that the UPPAAL semantics for broadcast channels is preserved. □

Proposition 5.7. *Consider a sequence* $\theta_{t_0}^{\mathcal{S}} \xrightarrow{*(+d)} \theta_{t_i}^{\mathcal{S}}$*, for some* $d \in \mathbb{N}$*, and lets assume* $\bar{l}_{t_0}^{\mathcal{S}} = \bar{l}$ *and* $w_{t_0}^{\mathcal{S}} = w$*. Then* $w_{t_i}^{\mathcal{S}} = w + d$*, and* $(\bar{l}, w) \rightarrow (\bar{l}, w + d)$ *is the only transition relation in the semantics of the UPPAAL model from* $(\bar{l}, w)$ *– i.e., there does not exist a transition* $(\bar{l}, w) \rightarrow (\bar{l}_1, w_1)$ *specified by either case (2) or (3) of Def. 5.2.*

Note that the chart's execution trace $\mathcal{R}^{\mathcal{S}}$ can be decomposed into a sequence of *SF-transitions*. Therefore, since the initial UPPAAL location vector $\bar{l}_0$ and valuation w_0 are equal to the initial Stateflow vector of active states and valuation $w^{\mathcal{S}}$, from the above three propositions and Theorem 5.1 we have that the sequence of *SF-transitions* for the UPP2SF-derived chart corresponds to an MPA run of the initial UPPAAL model. Furthermore, it is worth noting that these proofs can be easily extended to show that the semantics of committed locations is also preserved by the translation rules (semantics of urgent channels and locations are guaranteed by the *MPA-runs* requirement).

In the rest of the chapter, we will demonstrate the use of the UPP2SF-based MDD framework on the pacemaker case study.

5.7 Pacemaker Stateflow Design

From the model shown in Fig. 4.1, using the UPP2SF tool we obtained the pacemaker Stateflow chart presented in Fig. 5.3. For closed-loop verification in UPPAAL we modeled both the heart and pacemaker, and therefore the obtained chart contains both models of the controller (i.e., pacemaker) and environment (i.e., the heart). To be able to use the obtained Stateflow chart for both simulation and code generation it was necessary to decouple the pacemaker from the heart model.

Note that the verified UPPAAL model also contains several monitors used to specify verification queries. Since none of these monitors uses shared variables, and they only interact with the rest of the model by receiving synchronization over broadcast channels, they do not affect behavior of the basic automata from Fig. 4.1. Thus, to simplify Fig. 5.3, we did not show the parallel states that were obtained from them.

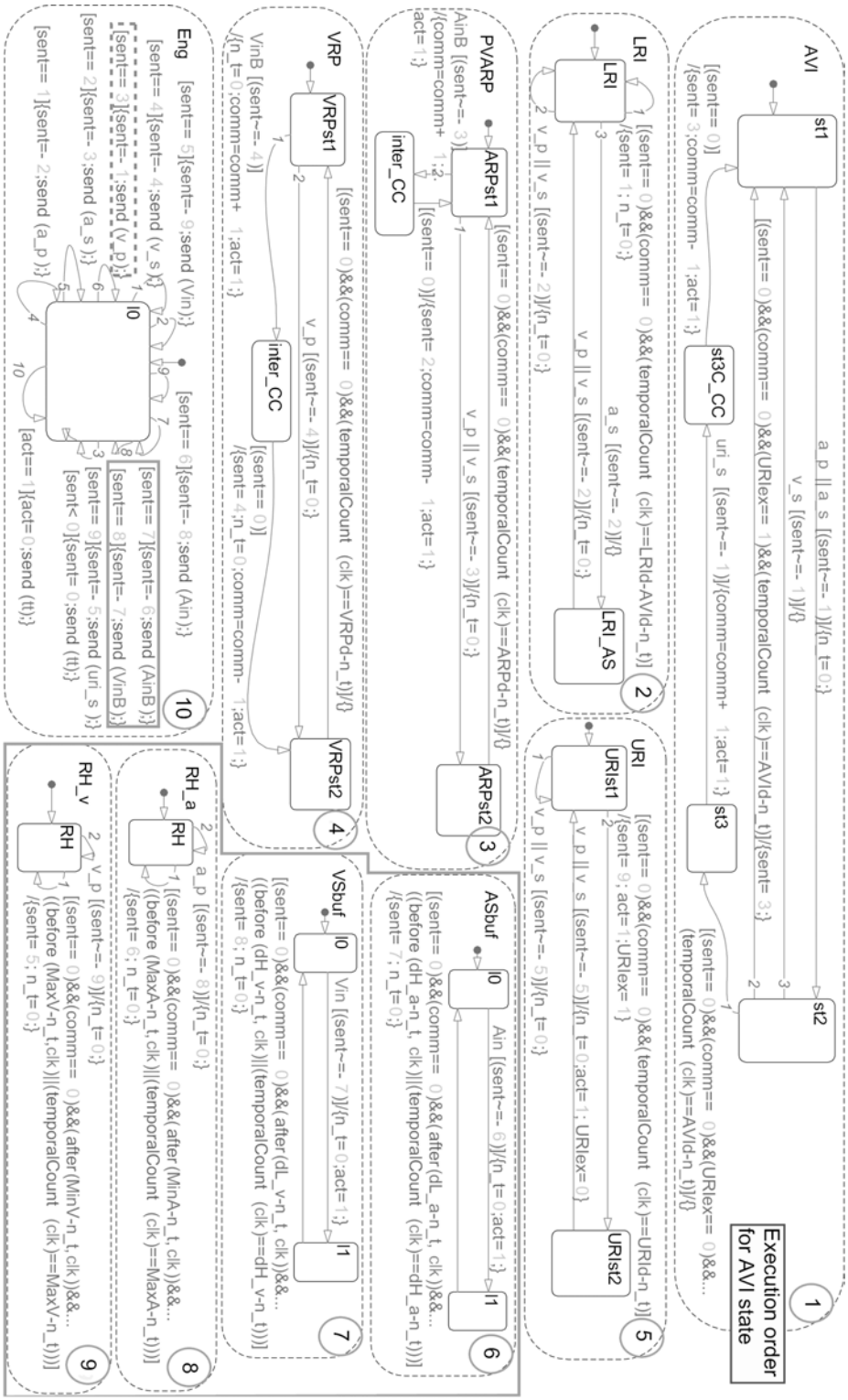


Figure 5.3: Pacemaker Stateflow chart extracted using UPP2SF from the UPPAAL model in Fig. 4.1; the heart and buffer models are highlighted.

5.7.1 Decoupling the Controller and Environment

Here we present the approach used to decouple the pacemaker and heart. The same approach can be used to decouple models of the environment and controllers in most commonly used scenarios where they only interact by broadcasting events.

Since the interaction between the heart and the pacemaker model is modeled using synchronization over broadcast channels, the pacemaker model can be easily extracted from the chart shown in Fig. 5.3. This is done by removing the parent states that model the heart and buffers (RH_a, RH_b, ASbuf, VSbuf), and by defining $AinB$ and $VinB$ as input events. Also, the Eng state has to be modified to remove the transitions used to broadcast these input events. In our case we removed the transitions that broadcast $AinB$ or $BinB$ (highlighted in red in Fig. 5.3).

Stateflow does not allow the use of output events to condition internal transitions. Hence, it is necessary to define additional output events from the chart, and in our case for local events a_p and v_p two output events (AP and VP) were defined. These events are broadcast on the same transitions used to broadcast a_p and v_p, respectively. In addition, to deal with some implementation issues (details are provided in Section 5.9), for each output `Event` an empty C function `sendHW_Event` is added using Simulink features for integrating custom C code. The function does not affect Simulink chart simulations, but allows for the correct output generation from the synthesized code. For example, the Eng transition highlighted with dotted green rectangle was modified to

$$[(sent == 3)]\{sent = -1; send(VP); sendHW\_VP(); send(v\_p);\}$$

Note that if the user specifies all components that are part of the controller, UPP2SF can automatically perform the above actions to decouple it from the environment.

It is interesting to compare the chart from Fig. 5.3 with the manually designed Stateflow model of the DDD pacemaker (Jiang et al. [2010b]), which is slightly simpler as it does not use event broadcasting. Thus, each *clk execution* has a single chart activation, causing a violation of several of the pacemaker requirements. For example, the URI requirement is not satisfied because URI state is always scheduled after AVI state; since the chart is not reactivated within a *clk execution*, after URI period expires, AP will be generated late – in the next *clk activation*.

Since the UPP2SF mapping has been validated, we ensure that the chart's execution will be equivalent to one of the MPA runs of the initial UPPAAL model. However, the chart also contains the model the environment and has no inputs and outputs, and thus we performed validation of the pacemaker Stateflow chart after the decoupling, by extending the approach for testing real-time constraints by Clarke and Lee [1995].

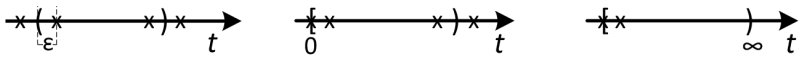


Figure 5.4: Test points for behavioral real-time constraints.

5.8 Stateflow Model Validation

We performed validation of the pacemaker Stateflow chart after the decoupling by extending the approach for testing real-time constraints described in Clarke and Lee [1995]. The same approach was also used for the testing of the final physical implementation (Section 5.10).

In Clarke and Lee [1995], the authors proposed a testing procedure for behavioral and performance constraints if the following assumptions are satisfied:

1. All output sequences of a system under test must be eventually distinguishable.
2. All time bounds must be constant.
3. There is no sharing of resources between concurrent threads.
4. Specification intervals are implemented in software as continuous, linear domains.

In our case, all of the assumptions are inherently satisfied, and thus we adapted the method for the pacemaker testing. From the formal specifications we derived an appropriate set of tests to validate correctness of the obtained pacemaker design. For each performance constraint we used a test that validates whether the appropriate output has been generated within the required interval. On the other hand, testing behavioral constraints was more complex. For each interval boundary we generated two tests. For closed boundaries we applied inputs that were exactly at the boundary point and tests that were outside the interval, at the distance ϵ from the boundary point (see Fig. 5.4). For open boundaries we generated inputs that were inside and outside the interval, at the distance ϵ from the boundary point.

To perform validation of the Stateflow chart and the physical implementation, we used the model parameters from Table 5.3. For testing in Simulink we considered only tests for the *ideal* system specifications. Since the chart was activated every 1ms, and transitions in Stateflow are instantaneous (all transition actions are atomic) we used $\epsilon = 0.5ms$ and $\epsilon = 1ms$ for simulations. All the 'ideal' real-time constraints (and thus, the constraints with tolerances) were satisfied in Simulink. This was expected since all actions within a *clk* execution are atomic to the event and no simulation time elapses during them.

In addition, the chart exhibited the same behaviors as the initial UPPAAL model. For example, for the aforementioned model parameters, when no inputs were applied the chart generated AP and VP pulses at the same time points as the UPPAAL model (i.e., AP were generated at $t_i^{ap} = (850 + 1000(i-1))ms,\ i = 1, 2, ...$, and VP at $t_i^{vp} = (1000i)ms,\ i = 1, 2, ...$). Similarly, no time was spent in committed states

$st3C\_CC$ in AVI, and $inter\_CC$ states in PVARP and VRP parallel states, and outgoing transitions from the states would occur immediately after the states were activated. To illustrate a more complex behavior we also showed that, as in UPPAAL, if $st3$ state in AVI parent state was active when the transition $URIst1 \rightarrow URIst2$ occurred (causing broadcast of uri_s event), then no time had elapsed in the $URIst2$ state, before the transition $URIst2 \rightarrow URIst1$ conditioned with v_p took place.

5.9 Pacemaker Implementation

We generated C code from the pacemaker Stateflow chart using the Simulink Real-Time Workshop Embedded Coder (RTWEC).[11,12] The code was generated for the general embedded real-time target and as a result we obtained the main procedure, `rt_OneStep`, which processes the three input events, $VinB$, $AinB$ and clk. To ensure that the model semantics is preserved (modulo the execution time), clk input events should be created every 1ms, followed by the procedure's activation.[13] This makes it suitable for implementation on top of a real-time operating system (RTOS).

Code Structure

The structure of the code is straightforward. The current state of the procedure and all variables defined in the chart are maintained in the structure `rtDWork`, along with counter values used for temporal logic operators (i.e., a counter per parallel state). In addition, `rtDWork` contains a structure (List. 1, Fig. 5.5) that for each parent state specifies if it is active, along with which of its substates is active. For example, for the state AVI variable `is_active_AVI` describes whether the state is active, while `is_AVI` specifies which of its exclusive substates is active.[14]

[11]Since Matlab R2011b, RTWEC toolbox is referred to as Simulink Embedded Coder.

[12]Although we focus on a specific implementation, code with the same structure would be generated from all Stateflow charts obtained from UPPAAL models using UPP2SF (due to the derived charts' structure).

[13]In this case, the procedure's execution corresponds to the clk execution.

[14]Note that since all parent states are decomposed into exclusive states, activity status for all of the substates within a parent state can be specified with a single variable. However, in the general case, if a state consists of a group of parallel states, RTWEC would define a new variable for each of the parallel states.

Parameter	Range	Value	Tolerance
LRI_d	343-1200 ms	1000ms	$\pm 4ms$
AVI_d	70-300 ms	150ms	$\pm 4ms$
URI_d	1000 ms	400ms	$\pm 4ms$
VRP_d	150-500 ms	150ms	$\pm 4ms$
ARP_d	150-500 ms	200ms	$\pm 4ms$

Table 5.3: Pacemakers parameters.

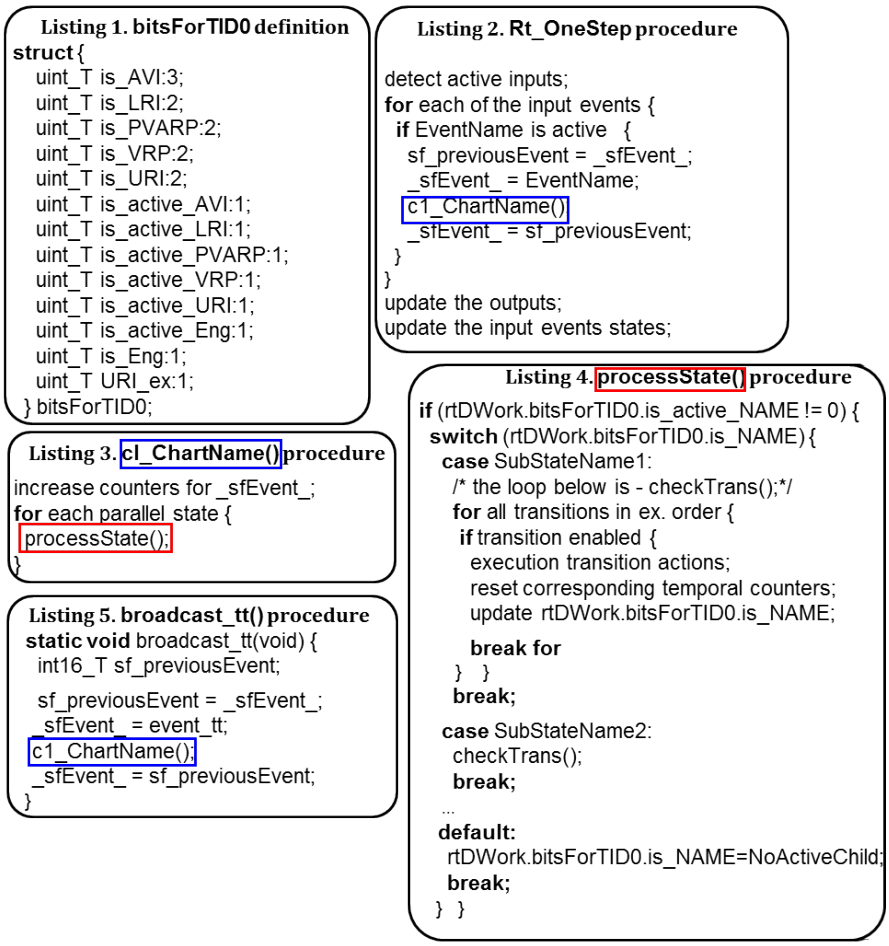


Figure 5.5: Structure of the pacemaker code obtained from the Stateflow chart shown in Fig. 5.3.

The structure of `rt_OneStep` is shown in List. 2, Fig. 5.5. After detecting active input events, an execution of the chart procedure `c1_ChartName` is invoked for each active input event. The variable *_sfEvent_* is used to denote the event that is processed during the chart execution. As in Stateflow, starting from input events with lower indices, the events are processed in a prespecified order (using `c1_ChartName` function). After all events are processed the procedure updates the outputs and event states in the prespecified order. This means that although we broadcast output events and the local events corresponding to them (e.g., VP and v_p) as a part of same transitions, the outputs will be actually updated at the end of `rt_OneStep` procedure. This can cause a couple of problems. First, ordering of the generated output events can differ from the order of the corresponding local events. Note that this does not affect simulations in Simulink, since all actions within a *clk execution* are atomic from perspective of the rest of the Simulink model. The second problem is that with this approach, for each output event only a single output trigger can be generated at the end of a *clk execution*. Thus, if an output event is broadcast more than once within a

single *clk execution*, the corresponding output events will be actually generated one by one, at the end of the consecutive *clk* executions (i.e., separated by the duration of *clk* period).

These issues are resolved using the aforementioned `SendHW_EventName` functions.[15] Using Simulink features for integrating custom C code with Stateflow charts in Simulink, we define empty C functions for each output event (e.g., for VP we define `SendWH_VP`). When the code is implemented on a particular hardware platform, the user needs to define these functions. For example, the simplest implementation would include toggling a particular CPU pin every time the function is invoked.

At the beginning of the chart execution procedure (List. 3, Fig. 5.5) all counters associated with the event (stored in $\_sfEvent\_$) are increased. Since the pacemaker code uses only *clk* event in temporal logic operators, the five counters will be incremented only when *clk* is processed. After this, the functions associated with each of the parallel states are called in the order specified by the execution order.

List. 4 from Fig. 5.5 presents a pseudo-code for processing each of the parallel states. If the state is active, all transitions outgoing from its active substate are evaluated in the prespecified execution order. The first enabled transition is taken and associated transition actions are executed. In the generated code only *Eng* state, which is executed last, is used to broadcast events as part of its transition actions. As shown in List. 5, Fig. 5.5, broadcasting an event associates the (current event) variable $\_sfEvent\_$ with the event, before it reactivates the chart (by calling `c1_ChartName()`).

5.9.1 Platform Implementation

The pacemaker code generated by the Simulink RTWEC was executed on nanoRK (Nano-RK [2007]), a fixed-priority preemptive RTOS that runs on a variety of resource constrained platforms. We tested the implementation on the TI MSP-EXP430F5438 Experimenter Board interfaced with a signal generator that provides inputs for the pacemaker code (Fig. 5.6). The compiled (without optimization) pacemaker procedure uses 2536 B for code and additional 180 B for data. To interface the code with the environment, each of the inputs ($AinB$, $VinB$) triggers an interrupt routine used to set the appropriate event for `rt_OneStep` function.

The pacemaker code was run as a task with period $1ms$. Table 5.4 shows measured execution times for the pacemaker tasks, for two different CPU frequencies. As expected, an increase in CPU frequency scales into a reduction in the task's execution time. Note that the measurements from Table 5.4 can be mapped to CPU utilization for the pacemaker task. With the average utilization of 9.2% for an 8MHz CPU, we can run multiple tasks on the RTOS.

[15] These issues do not present a problem for the pacemaker design from Fig. 5.3, since only a single AP or a single VP can be broadcast within one *clk execution*. However, in this chapter we describe the general approach that allows utilization of the UPP2SF translation tool for all UPPAAL models.

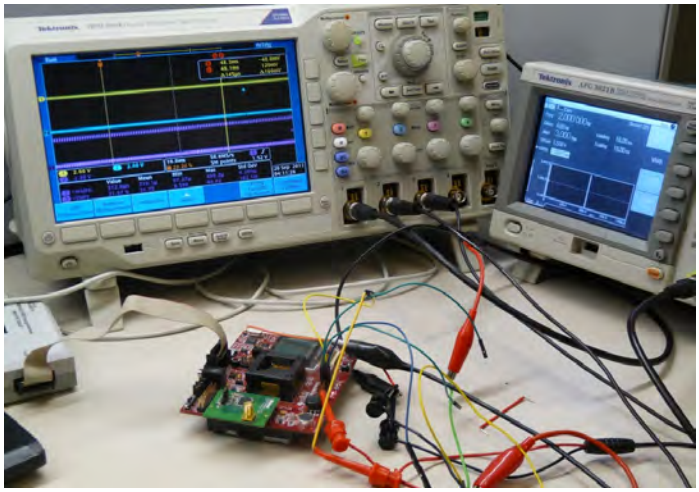


Figure 5.6: Hardware setup with MSP430F5438 experimenters board.

5.9.2 Decoupling the Controller and the Environment

In Sec. 5.7 and 5.9.1 we have described the method we used to decouple models of the pacemaker and the heart. The solution guarantees that the implemented code generates output events as soon as the corresponding local events are generated. However, our implementation introduces some problems regarding processing of input signals. By introducing an interrupt routine that sets a flag if the input occurs, we effectively synchronize asynchronous input signals. This has a twofold effect on the implemented code. First, each input signal will be processed at most once even if it appears more than one time between consecutive task's activations. This is not a problem for the pacemaker, since in the initial UPPAAL model, due to the buffers, all inputs after the first input in a cycle are disregarded until at least dL_a (or dL_v) time. Second, it introduces a latency up to the task's period (i.e., *clk* interval, in our case 1ms) before the input signals are processed. To solve this issue, the extracted procedure could be activated as soon as an input appears. Beside problems with tasks scheduling, if the input signal could affect clock valuations in the initial UPPAAL model this would introduce a time measurement error in the code. For example, if *Ain* occurs 0.5ms before the next *clk* activation and the procedure is instantaneously activated as a result of the input, the clock in $ASbuf$ would be reset to zero. Thus,

CPU frequency	Average ex. time	Minimal ex. time	Maximal ex. time	Standard deviation
4MHz, OL	176.1μs	167.6μs	462.9μs	14.2μs
4MHz	180.9μs	167.6μs	738.2μs	17.3μs
8MHz, OL	89.5μs	84.7μs	234.6μs	7.2μs
8MHz	92.0μs	84.9μs	370.4μs	13.7μs

Table 5.4: Execution times for the pacemaker procedure; OL denotes open-loop, without inputs from the signal generator.

the next *clk* activation of the procedure would set t to 1, although only 0.5 ms have passed since the input.

This problem occurs even if the code has been generated using Times, or any other tool, since the number of clocks used in models is usually greater than the number of timers that CPU provides. To avoid this type of errors, we opted to use the aforementioned approach where input events only set a flag to indicate the need to process input events in the following procedure activation. To take this into account in the initial UPPAAL model, we reverified the safety properties for the model where input buffers increase the upper bound on the introduced delay (i.e., dH_a, dH_v). The bounds are increased to incorporate the maximal input latency introduced by synchronous processing of the input events (in our case 1ms).

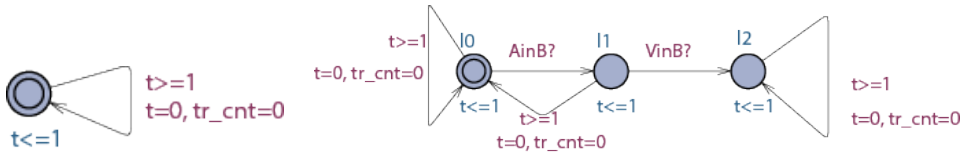
5.9.3 Worst Case Execution Time Estimation in UPPAAL

Correctness of the generated code relies on the assumption that execution of the code completes before the next external activation. To make sure that it does, we need to estimate the WCET of the code execution, taking into account that the `c1_ChartName` procedure (i.e., the chart) may be internally activated multiple times. We propose an approach that does not require translation from UPPAAL to Stateflow. Rather it uses the initial UPPAAL model to calculate an upper bound on the maximal number of internal activations N_i within an external activation (i.e., per *clk* execution). This enables a WCET estimation at an early stage, during system modeling in UPPAAL.

Since the chart is reactivated with event broadcasts and some transitions, to determine the bound for N_i we extend the model with the following accounting features:

- Global variable tr_cnt and the automaton `TrMonitor` (Fig. 5.7) that resets the variable at integer time points,
- In the controller part of the UPPAAL model, reset operation $tr\_cnt = tr\_cnt+1$ should be added to all edges with transmissions over a broadcast channel, or edges that would be translated into Stateflow transitions with $act = 1$ reset (i.e., the transition for which *Eng* state would reactivate the chart),
- Reset $tr\_cnt = tr\_cnt+1$ to the edges with transmissions over broadcast channels that present inputs to the controller,
- Introduce UPPAAL temporal formula $A\Box\ tr\_cnt \leq \tilde{N}_i$.

With the above changes the variable tr_cnt bounds the number of internal activations of the chart. Therefore, if the above proposition is satisfied, the value $\tilde{N}_i + 1$ provides an upper bound for the number of chart executions within a single *clk* execution (1 is due to external activation). For the pacemaker UPPAAL model from Fig. 4.1 we added the reset operation to 8 transitions. We proved that the formula



(a) Basic monitor (b) Transition automaton with input ordering

Figure 5.7: Transition monitors (`TrMonitor`) used for the worst-case execution time estimation.

holds for $\tilde{N}_i = 5$. Note that since the UPPAAL model contains a model of the environment (i.e., the heart), $\tilde{N}_i$ takes into account chart activations caused by inputs. On the other hand, when we considered open-loop execution of the pacemaker (without input events from the heart), using the pacemaker model Fig. 4.1 without the model of the environment (i.e., RH automata) we proved that the formula holds for $\tilde{N}_i^{ol} = 1$. Thus, in this case at most two chart executions can occur within a single *clk* execution (i.e., task activation).

If these results are compared with the execution time measurements from Table 5.4, we can notice that for the open-loop experiments, the ratio between the maximal and minimal execution time is less than 3. Similarly, for the experiments with the test generator, the ratio is less than 5. Since in our case the minimal execution time corresponds to a single chart execution during the task's activation (which in general might not be the case), we can infer that $N_i^{ol} = 1$ and $N_i = 3$. Therefore, our WCET analysis provided the exact bound for the open-loop scenario and a conservative bound for the closed-loop case.

The reason for this is that the transition monitor from Fig. 5.7(a) does not take into consideration the order of the input events processing, and if both $AinB!$ and $VinB!$ occur at the same time instance, the UPPAAL model might synchronize over the channel $AinB$ first. On the other hand, the pacemaker model in Stateflow, and thus the obtained code, have a fixed input ordering, meaning that the inputs are always processed in the predefined order; in the pacemaker code $VinB$ is always processed before $AinB$ (and the *clk* event is processed last). To take this into account we used the monitor from Fig. 5.7(b) and specified the proposition as $A\Box\ ((tr\_cnt \leq \tilde{N}_i)\ ||\ (TrMonitor.l2))$. This effectively disregards scenarios in which $AinB$ is processed before $VinB$ within a task execution. We proved that this formula holds for $\tilde{N}_i = 4$, thus improving the bound. Note that if the obtained code has more than two inputs (beside *clk*) it is necessary to specify all possible invalid combinations of the inputs' ordering, which might significantly increase the verification time.

5.10 Testing of the Physical Implementation

We validated the physical implementation using the procedure from Section 5.8. Unlike validation of the Stateflow chart, for physical testing we considered two types

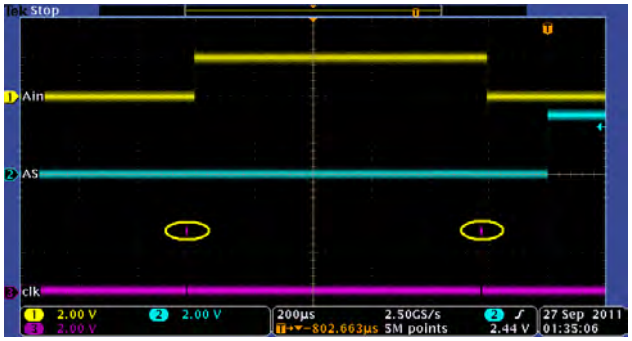


Figure 5.8: A test screen shot for property **B4.2**; *clk* pulses are highlighted.

of tests. For the *ideal* system specifications we used $\epsilon \leq 80\mu s$, since $84.9\mu s$ was the chart's minimal execution time (Table 5.4). Similarly, since the values for all the predefined tolerances are $\pm 4ms$, for the second set of tests we used $4ms < \epsilon \leq 4.08ms$.

Table 5.5 presents testing results for the pacemaker implementation executed on the MSP430 Experimenter Board. When the tolerances are not taken into account some of the properties that were verified in UPPAAL and validated in Simulink were violated during the tests. The reason is that the UPPAAL semantics uses an unrealistic assumption that the machine executing the code is infinitely fast (i.e., no time elapses during transitions) and the system's reaction to synchronization is instantaneous. In the general case, the execution delays can cause violation of the UPPAAL semantics in the obtained physical implementation, which is the main reason for violation of some of the verified safety properties. However, when interval tolerances are taken into account, all properties were satisfied, as shown in Table 5.5.

For example, consider the property **B4.2**. Fig. 5.8 presents one of the oscilloscope screenshots obtained during the testing. The signals shown are *Ain* (top), *AS* (middle) and *clk* (bottom). As shown, *Ain* appeared right after the first *clk* occurrence. It sets the appropriate flag in the interrupt routine, but the processing of the corresponding event occurred with the next *clk*. The event processing takes approximately $232\mu s$ before AS is generated. This, along with the time (up to $1ms$) between *Ain* and the following *clk*, results in delay of up to $1.232ms$. Thus, *ideal* requirement **B4.2** is violated. However, since the delay is within the tolerance bound, the requirement is satisfied when the tolerances are taken into account.

Requirement	P1.1	B1.1	B1.2	P2.1	P2.2	B2.1	B2.2
Ideal	Pass	Fail	Pass	Pass	Pass	Fail	Fail
With tolerance	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Requirement	P3.1	B3.1	B3.2	P4.1	B4.1	B4.2	
Ideal	Pass	Fail	Fail	Pass	Fail	Fail	
With tolerance	Pass	Pass	Pass	Pass	Pass	Pass	

Table 5.5: Results of the tests performed on the setup from Fig. 5.6.

5.11 Discussion

We have described the design of the UPP2SF tool for automatic translation of UPPAAL models in Stateflow. We have shown that for a large class of UPPAAL models, UPP2SF preserves behavior of the initial UPPAAL model. Furthermore, we have presented an UPP2SF-enabled Model-Driven Development framework for safety-critical system design. By applying the UPP2SF model translation tool on the dual-chamber implantable cardiac pacemaker case study, we have demonstrated the process starting from the formalization of the device specifications, followed by system modeling and verification in UPPAAL, to closed-loop system simulation in Simulink/Stateflow and testing of the physical implementation. We have also shown how the translation tool provides a way to estimate WCET during modeling and verification stage in UPPAAL, and facilitates development of modular code from UPPAAL timed-automata based models.

However, UPPAAL as a design tool cannot be used to design more complex systems. For instance, in UPPAAL the value of the clocks cannot be saved. Systems that require timing information as history cannot be explicitly modeled in UPPAAL, which also limits the application of UPP2SF.

Chapter 6

Theme 4: in-silico Pre-clinical Trials for Implantable Cardiac Devices

10,000 people in the U.S. receive an Implantable Cardioverter Defibrillator (a heart rhythm adjustment device) every month (ICD [2015]). Clinical trials have presented evidence that patients implanted with ICDs have a mortality rate reduced by up to 31% (Moss et al. [2012]). Unfortunately, ICDs suffer from a high rate of *inappropriate therapy*, which takes the form of unnecessary electric shocks or pulse sequences delivered to the heart. Inappropriate therapy increases patient stress, reduces their quality of life, and is linked to increased morbidity (Rosenqvist et al. [1998]). Depending on the particular ICD and its settings, the rates of inappropriate therapy range from 46% to 62% of all delivered therapy episodes (Gold et al. [2012]).

After the verification and testing effort is completed, regulatory agencies like the F.D.A. require that the safety and efficacy of new devices be demonstrated in a *CT* (Fig. 6.1). In a trial, a group of patients that are treated with the new device (this is the 'intervention group') are compared to a group of patients who are treated with the current standard of care (e.g., a different device currently on the market; this is the 'control group'). The objective is to see whether the different devices result in significantly different effects on the patients. Clinical trials are major endeavors, involving physicians, patients, statisticians, clinical centers, companies and regulators, sometimes in several countries. Late-phase trials can run for several years, and cost millions of dollars. For example, a 2002 trial for stents lasted 2 years, enrolled 800 patients and cost $10 to $12 million and lasted 24 months (Kaplan et al. [2004]). Trials also pose an inherent risk to the patients in the intervention group by exposing them to an unproven device. Thus it is crucial that they be well planned, and rigorously executed.

In reality, any trial runs the risk of errors during its planning and execution stages, which can invalidate the results of the trial. In this paper, we pose and propose an answer to the following question: *how can modeling of CPS assist in the planning*

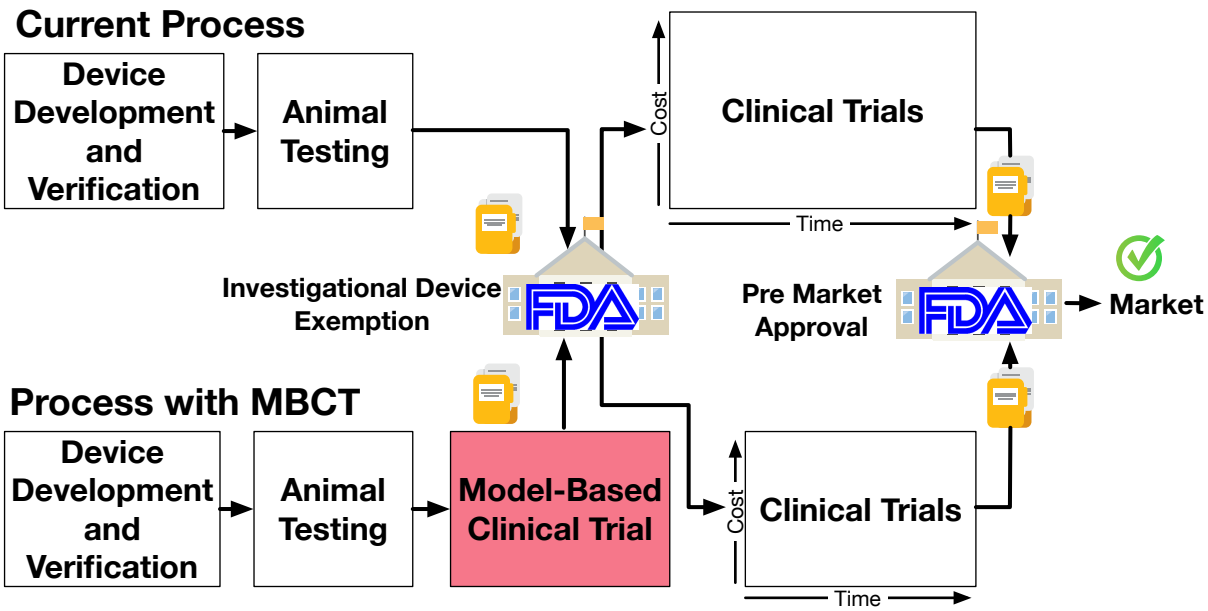


Figure 6.1: Bringing a device to market. Clinical trials are the last step before a new device's market approval. Model-based clinical trials will provide insight during planning and execution of clinical trials, leading to reduction in costs and increasing the chance of a successful trial.

and execution of a clinical trial, so as to increase the chances of a successful trial?

Most medical models today are aimed at either better understanding the phenomenon under study (Ten Tusscher et al. [2007]) or at device debugging and verification (Jiang et al. [2012a]). There is only one case in which a computer model has been used to intervene in the regulatory process of medical devices, namely the T1 Diabetes Model (T1DM) of UVA/PADOVA (C. Toffanin, M. Messori, F. Di Palma, G. De Nicolao, C. Cobelli, L. Magni [2014]). T1DM models glucose kinetics in hypoglycemia, and has been accepted by the FDA as a substitute for animal trials. The T1DM has a fixed virtual cohort with 300 patients. Its objective is to test the efficacy of new glucose control algorithms by simulating them on the virtual cohort. While our models can be used in this way, our objective here is to target specific clinical trials steps and improve how they are conducted. This dictates the experimental setup and the cohort generation considerations.

The Avicenna consortium (Avicenna Consortium [2015]) lays out a vision for In-Silico Clinical Trials' similar to our approach. However, the emphasis in Avicenna is on individualized patient models, as they propose to customize the model to each patient enrolled in a trial. In the present work, we propose a usage of in-silico pre-clinical trial *prior* to recruitment. Thus our models need not be fitted to a given patient's data, which might be impossible, invasive, or burdensome for the conduct of the trial.

In this chapter, we demonstrate how computer models can be used for early,

affordable and reproducible testing of a clinical trial's premises and assumptions. Model-based empirical validation of the premises reduces the risk of conducting a trial that fails to demonstrate the desired effect (typically, an improvement of new intervention over the control). We used the Rhythm ID Going Head to Head Trial (RIGHT) (Gold et al. [2012]), which lasted five years and sought to compare the diagnostic algorithms used by two ICDs for correctly diagnosing potentially fatal tachycardias (abnormally fast heart rhythms).

6.1 Clinical trials and RIGHT

At the clinical trial stage (Fig. 6.1), the objective is no longer to find bugs in the device: it is, rather, to evaluate the safety and efficacy of the validated device on humans. RCT are the gold standard for evaluating the safety and efficacy of a new medical device (L. M. Friedman and C. D. Furberg and D. L. DeMets [2010]). They constitute the only time prior to market use where the effects of the device on humans are actually observed, and are legally mandated for new high-risk medical devices like ICD. The planning and execution of an RCT requires carefully navigating a number of technical, logistical and ethical issues to obtain reliable and statistically significant results.

Because of the very high cost of RCT in terms of money, time, and the risk of harm they present to enrolled patients, our focus in this paper is on the use of CPS models, formalized in an in-silico pre-clinical trial, *to validate the assumptions made by the investigators and thus increase the chances of success of an RCT*. We illustrate our approach by applying it to the Rhythm ID Going Head-to-Head Trial (RIGHT) (Gold et al. [2012]), which we present next.

6.1.1 The RIGHT trial

We first provide a brief background to better understand RIGHT. Tachycardias (abnormally elevated heart rates) can be divided into VT, which originate in the heart's ventricles, and SVT, which originate above the ventricles. A sustained VT can be fatal, while an SVT is typically non-fatal. The therapy applied by the ICD often takes the form of a high-energy electric shock. The shock can be pro-arrhythmic, and was even linked to increased morbidity (Rosenqvist et al. [1998]). Therefore, one of the biggest challenges for ICDs is to guarantee shock delivery for VT, and simultaneously reduce inappropriate shocks during SVT (Ellenbogen et al. [2011]).

RIGHT is a trial that sought to compare the VT/SVT discrimination abilities of two algorithms (Gold et al. [2012]): the Rhythm ID detection algorithm found in Boston Scientific's Vitality II ICDs (Boston Scientific Corporation [2007b]), and the PR Logic + Wavelet (PRL+W) detection algorithm found in a number of Medtronic's ICDs (Medtronic Maximo, Marquis, Intrinsic, Virtuoso, or Entrust ICD). *Inappropriate therapy* was defined as therapy applied to an arrhythmia other than VT or VF

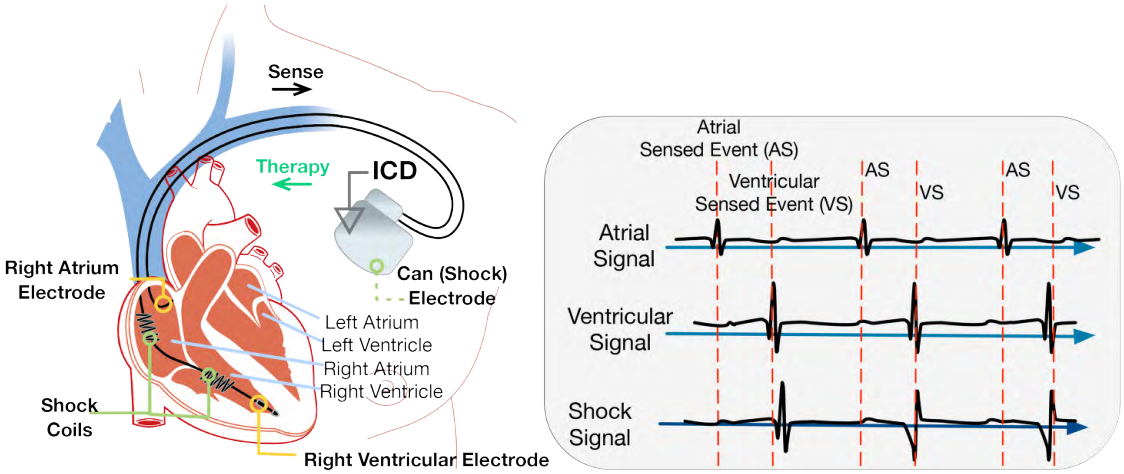


Figure 6.2: ICD connected to the heart. The atrial, ventricular, and shock electrogram signals are measured by the device, which uses them to diagnose the current state of the heart and determine whether therapy is required.

(VF is a type of VT). RIGHT enrolled 1962 patients and ran for approximately five years. It was fully sponsored by Boston Scientific.

One of the trial's assumptions was that Rhythm ID would reduce the risk of inappropriate therapy by 25% over PRL+W (Berger et al. [2006]). The outcome of the trial (Gold et al. [2012]), however, was that patients implanted with ICDs running Rhythm ID had a ***34% risk increase*** of inappropriate therapy as compared to patients implanted with ICD running PRL+W. This result is the opposite of the effect hypothesized by the trial investigators. In this paper, we design an in-silico pre-clinical trial to test early and quickly whether the hypothesized effect holds by comparing the two ICDs on a large *synthetic* cohort.

Organization: In the following sections we describe the building blocks of in-silico pre-clinical trials: modeling the heart, processing 100's of real patients' data, mapping the timing and morphology components of the signal to a heart model we developed, generating a population of 10,000+ synthetic heart models, implementing the device algorithms and conducting multiple trials for the comparative rate of inappropriate therapy, condition-level rates and evaluating the effect of device parameters on discrimination rates.

6.2 Virtual Cohort Generation

Implantable Cardioverter Defibrillators (ICDs) can diagnose VT and VF by observing the electrical activity through three channels, as shown in Fig. 6.2. The measured signals are known as *electrograms*, or EGMs. VT and SVT can share similar heart rates and might even occur simultaneously, so an SVT can be mis-diagnosed as a VT. This is problematic because VT therapy consists of low and high energy electric shocks of 30-40 Joules (∼800V) delivered directly to the heart, which is very painful to the patient, and has been shown to increase morbidity (Rosenqvist et al. [1998])[1]. Therefore, one of the biggest challenges for ICDs is to discriminate between VT that typically requires a shock, and SVT that typically should not be shocked (Ellenbogen et al. [2011]).

An EGM signal can be characterized by the *timing of events* that produced it, and the *morphology of the signal itself*. An 'event' is roughly characterized as the source of the largest peak in the EGM (e.g. a ventricular depolarization), and event timing is a crucial element of an arrhythmia's definition in clinical Electrophysiology. The 'morphology' refers to the shape of the EGM (see Fig. 6.5 for examples). Both aspects are used by the ICD to make its decision. Correspondingly, our model has two components: a timing model, and a morphology model.

6.2.1 Timing Model

The node and path topology used in the in-silico pre-clinical trials is shown in Fig. 6.3. The hollow nodes are passive nodes representing key locations within the heart where electrical events may be blocked. These include the Atrioventricular node (AV), Right Bundle Branch (RBB) and Left Ventricle Apex (LVA). The filled nodes in red, Sinoatrial (SA) node and Right Ventricle Apex (RVA) node, represent the heart locations where ICD electrodes are placed to measure the EGMs. The timing of the activation events at these nodes determines the timing of corresponding EGMs. Different sources for tachyarrhythmias are represented by arrhythmia nodes (dashed filled nodes) which are capable of self-activating at prescribed rates. These include Premature Atrial Complexes (PACs) and Premature Ventricular Complexes (PVCs) which are sources of rhythm disturbances.

Every node and path automaton has timing parameters that determine, for example, the delay between events, and how long it takes to conduct an electrical event between two nodes. These timing parameters can be directly derived from clinical data (Josephson [2008]), and the model structure is compatible with clinical Electrophysiology concepts. Thus we know the ranges for these parameters. In Jiang et al. [2012a], the timing model's capability to simulate various normal and abnormal heart conditions was validated quantitatively and by cardiac electrophysiologists.

In this work we use the same heart model structure to ensure the correct timing

[1]Physicians compare a shock to a "horse kicking you in the chest"

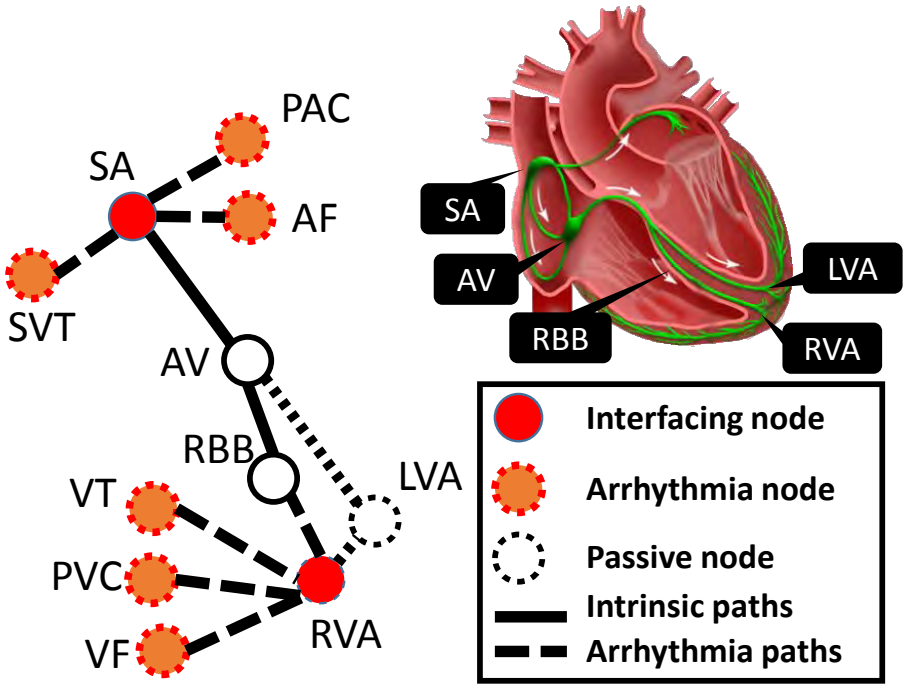


Figure 6.3: Timing model of the heart

of the EGM signals into the ICD. In order to account for inherent timing variability, during simulation the heart model randomly selects timing parameters within a pre-specified range, instead of choosing specific values. By choosing the range, we control the variability of the signals produced by a given model instance.

6.2.2 Morphology Model

The ICD uses the EGM morphology in two ways: first, the atrial and ventricular EGMs are used to *sense* when events occur via peak detection (Section 6.3.1). Second, the Shock channel EGM is used in the morphology comparison discriminators (Section 6.3.2, Gold et al. [2002], C. D. Swerdlow et al [2002]). It is known that sensing (the detection of events) can be responsible for up to 20% of inappropriate therapies (Daubert et al. [2008]). Therefore, it is important that our model generate realistic and varied EGM waveforms for a proper evaluation of the detection algorithms.

The timing model provides the time stamps for electrical events to happen at the interfacing nodes (SA,RVA). From path conduction we also know the source of the signals. In the heart model structure shown in Fig. 6.3 there are 5 different sources for SA node activation and 5 different sources for RVA node activation. Based on the clinical observations that electrical events from the same source produce very similar EGM morphologies, we can generate EGM signals by overlaying EGM templates corresponding to different sources onto the timing event diagram. The procedure is shown in Fig. 6.4. We also introduce small variations on EGM templates. The

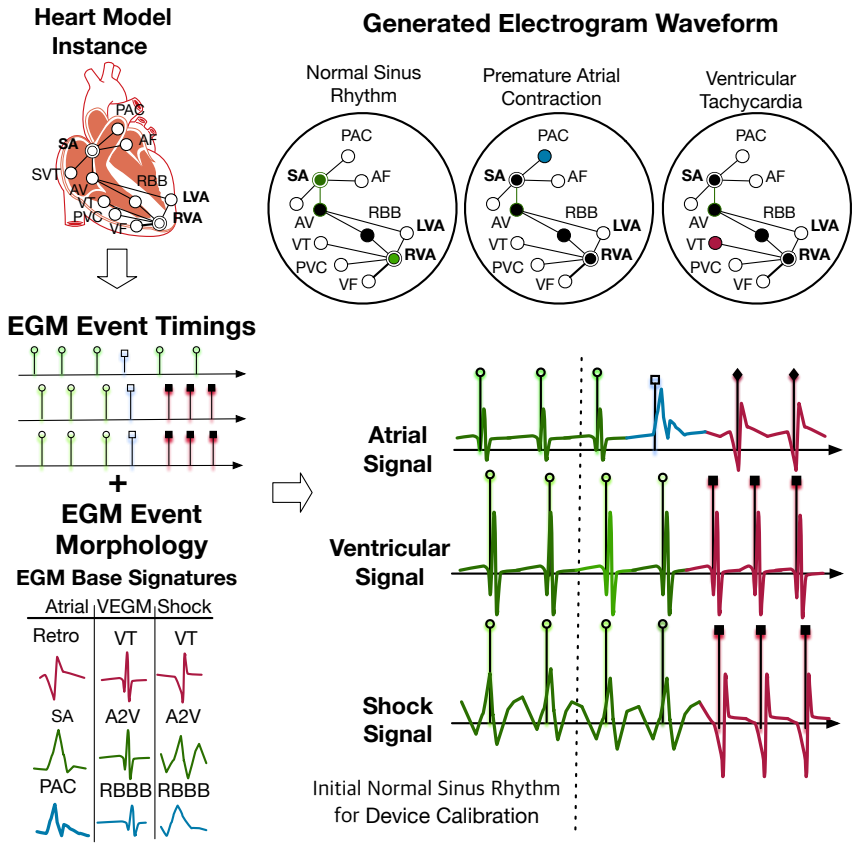


Figure 6.4: EGM waveform generation. From a given model instance and set of tachycardias, an EGM waveform is generated for the duration of an episode. The timing model determines event timings. When an event occurs, the EGM morphology for the event is output from the morphology model.

variations are obtained by a wavelet decomposition of the signatures followed by a random scaling of the 25% smallest coefficients. We guarantee that this does not change the signature of the EGM, by running one of the morphology comparison discriminators described in Section 6.3.2. This variation is parametrized, e.g. the percentage of modified coefficients, the range of the random scaling.

6.2.3 Patient Data Adjudication and EGM Template Extraction

In order to obtain realistic morphologies for our simulations we utilize the Ann Arbor Electrogram Libraries (AAEL), a database of over 500 EGM recordings made during clinical electrophysiology studies (Jenkins and Jenkins [2003]). The AAEL is used by all major ICD manufacturers and is licensed by the US FDA. The AAEL provides descriptive annotations of records at a high level. We performed additional detailed examination to precisely segment each record according to rhythm type.

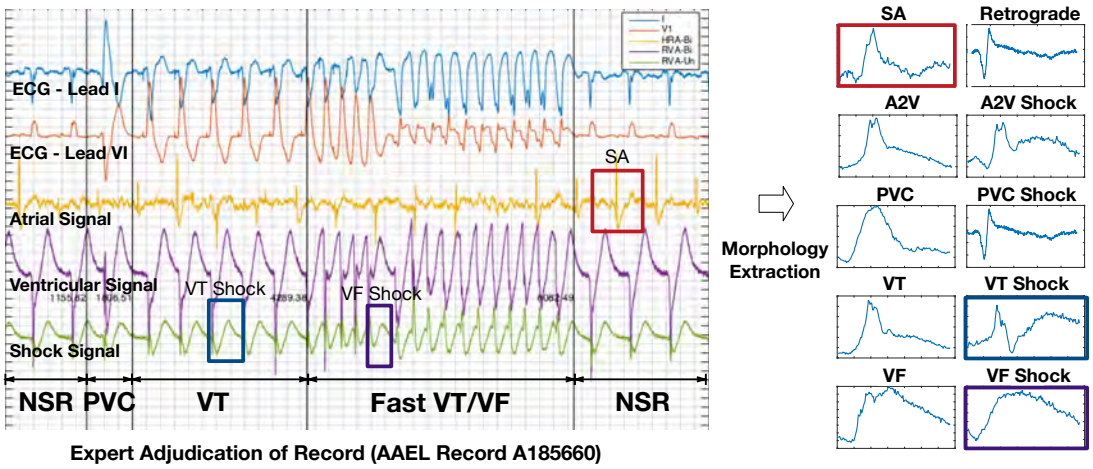


Figure 6.5: (Left) The EGM record is segmented into episodes with distinct rhythms in each. (Right) From each episode, individual EGM morphologies are extracted and stored.

123 records from 47 patients were manually examined and adjudicated into segments called *episodes* containing one specific rhythm, e.g. NSR or VF. The adjudication was performed by a cardiologist. Fig. 6.5 (left) shows an example record (Record A185660) which has undergone this adjudication. From each episode, we developed an automated process which extracted EGMs from a given episode. The EGM are collected and organized by both patient record and by the type of rhythm which was annotated during the adjudication process. These extracted rhythm *signatures* provide the basis for the morphology information in the signal generated by our model. Fig. 6.5 (right) depicts an example of 10 signatures extracted from the record.

6.2.4 Cohort generation

Let $p = (p_1, \ldots, p_n) \in \Re^n$ be the vector of timing and morphological parameters of the heart model. Let $P_i \subset \Re$ be the range of parameter p_i. We generate a *synthetic cohort* of N probabilistic model instances. To produce one of these instances, for each scalar parameter p_i, we randomly select a sub-interval I_i of its range: $I_i \subset P_i$. The sub-interval I_i is chosen so that it fits with the tachycardia that this model instance is meant to simulate. E.g., for modeling VT, the rest period of the VT node might be assigned the sub-interval $I_i = [260, 280]ms$, reflecting the firing rate in the ventricles. When a model instance is simulated, each parameter p_i's value changes beat to beat by sampling it uniformly within its sub-interval I_i. Thus each generated model is probabilistic to reflect inherent rhythm variability.

6.3 Implementing Device Algorithms

Due to the limited sensing capability of ICDs, device manufacturers have developed different algorithms to identify the electrical events and correctly diagnose the cardiac arrhythmia as being VT or SVT. In this chapter we implemented the detection algorithm Rhythm ID of Boston Scientific (Boston Scientific Corporation [2007b], Ellenbogen et al. [2011]),and PRLogic+Wavelet (PRL+W) of Medtronic (Singer [2001], C. D. Swerdlow et al [2002]). We also set up a testing platform to validate our implementations against real ICDs using conformance testing.

6.3.1 Cardiac Signal Sensing

Sensing is the process by which cardiac signals measured through the leads of the ICD is converted to cardiac timing events. Appropriate sensing is essential for proper ICD detection algorithm operation which relies heavily on accurate event timing and morphology information provided by sensing. An *event* corresponds to a depolarization in the heart and manifests as a displacement from the baseline amplitude of the signal. In its simplest form, the sensing algorithm declares an event whenever the amplitude of the signal exceeds a given threshold. Once an event has been declared, the peak of the amplitude is measured and a *refractory period* begins, during which a consecutive event is ignored for a short period of time. This is to ensure that the same event is not counted repeatedly.

ICDs require a balance in sensitivity in order to operate in noisy, complex, environments where cardiac events can vary greatly in signal amplitude and frequency, such as during VF. Setting the threshold low achieves higher sensitivity to events of small amplitude, but increases the chances for incorrectly sensing other cardiac electrical artifacts such as T-waves and noise artifacts (oversensing). Conversely, setting the threshold too high allows sensing to be more robust to noise and other cardiac electrical events, but creates the potential for undersensing events of interest, such as during VF when the peak amplitude can be low.

In order to achieve adequate balance, ICD sensing algorithms are enhanced by applying dynamic adjustment of the sensitivity threshold. Initially, the threshold is raised during the refractory period after an event and once the refractory period concludes, the threshold is decayed to a minimum pre-set threshold. In our device models, we implemented the AGC algorithm of Boston Scientific and the AAS of Medtronic ICDs to incorporate dynamic threshold adjustment.

The complexity of these enhancements adds to the difficulty of properly programming device settings of ICDs and requires calibration process at the time of ICD implantation and during patient follow-up visits.

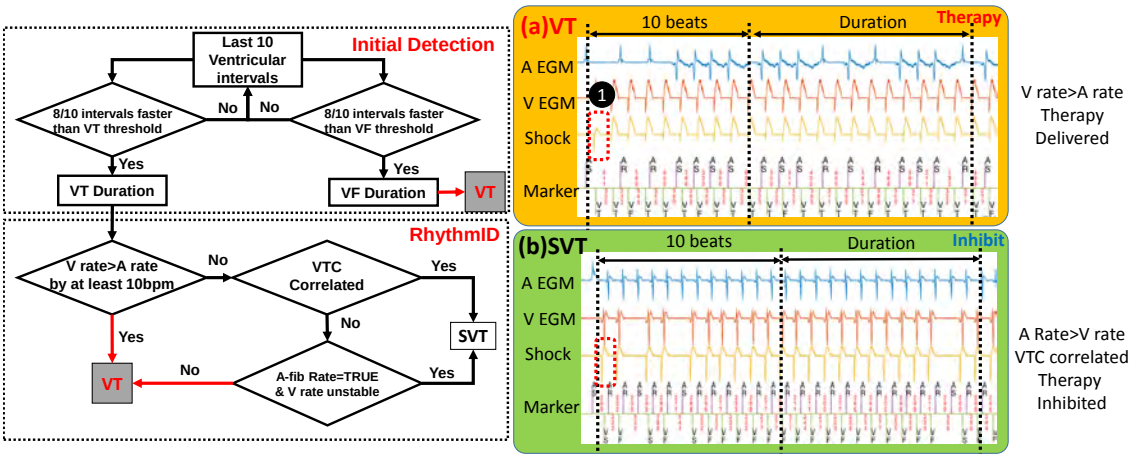


Figure 6.6: SVT/VT detection algorithm by Boston Scientific (Boston Scientific Corporation [2007b]). The two cases on the right illustrate two different decisions by the algorithm. (a) illustrates a sustained VT case where at the end of the Duration, the ventricular rate is faster than the atrial rate. The algorithm correctly identified the rhythm as VT and delivered therapy. (b) illustrates a SVT case where at the end of the Duration, the ventricular rate is slower than the atrial rate. Then by comparing the EGM morphology in the Shock channel (Marker 1) with the stored NSR template (Marker 2) for the last 10 EGM events, the algorithm decided that the morphology is correlated, therefore therapy is inhibited.

6.3.2 VT Detection Algorithm

Device companies have developed different algorithmic components to distinguish SVT from VT, referred to as *discriminators*. Each discriminator utilizes the history of timing and/or morphology of the EGM signals to determine whether the current rhythm is a VT or SVT (or neither). No single discriminator is sufficient on its own to discriminate between SVT and VT, because these classes of arrhythmias can appear similar in a number of criteria. Therefore discriminators are organized in a decision tree.We have implemented the detection algorithms Rhythm ID from Boston Scientific and PRL+W from Medtronic. This section gives an overview of both algorithms.

Rhythm ID

Rhythm ID's decision tree is shown in Fig. 6.6. Rhythm ID detects an episode by continuously examining the last 10 ventricular intervals and comparing them with VT and VF thresholds. If 8/10 intervals are shorter than the VF threshold for a certain pre-set *VF Duration* (e.g., 2.5 seconds) then the algorithm declares VF. Otherwise, if 8/10 intervals are shorter than the VT threshold for a certain VT Duration, then further discriminators are used. First, if the ventricular rate is greater than the atrial rate for the last 10 ventricular beats, Rhythm ID will determine the condition is VT. Otherwise, the Vector Timing and Correlation (VTC) discriminator (Gold et al. [2002]) compares EGM morphology of the last 10 ventricular events with an EGM *template* saved during NSR. VTC is based on the assumption that the EGM

morphology of the shock channel during VT is different from its morphology during SVT and NSR. If the correlation between the current EGM's morphology and the stored NSR morphology is above a pre-set threshold, the current rhythm is more likely to be SVT than VT, and therapy is withheld. Otherwise, if the atrial rate is equal to a pre-set fibrillation rate and the variance of the ventricular interval length exceeds a certain limit, the algorithm decides it's an SVT. Otherwise, it decides this is a VT.

PR Logic+Wavelet

PRL+W also utilizes rate-based and morphology-based discriminators similar (but not identical) to Rhythm ID. The morphology discriminator used in PRL+W is similar to VTC in Rhythm ID, but operates in the wavelet domain (C. D. Swerdlow et al [2002]). PRL+W also continuously compares the pattern of atrial and ventricular activation to 19 pre-defined patterns (Singer [2001]). Each pattern is associated to a heart condition, like Sinus Tachycardia or VT. A match between the current activity and one of the pre-defined patterns is used as an indication that the current rhythm is explained by the associated condition.

6.3.3 Validation

Conformance testing was used to validate the software implementations of the Vitality II device by Boston Scientific. The validation hardware setup is illustrated in Fig. 6.7. 14 different scenarios were specified and programmed into an EGM Waveform generator (CRM3 Simulator, Guidant, USA) such that it would output a signal to the connected Vitality II device. The various scenarios traverse 7 out of 9 branches of the detection algorithm for Boston Scientific described in Sec. 6.3.2 and shown in Fig. 6.6. The response of the ICD interrogated using an ICD programmer (ZOOM Latitiude, Boston Scientific). As the waveform was applied to the ICD, the waveform was simultaneously acquired using a National Instruments DAQ board. The recorded waveform was then applied to the device model and response was compared. Fig. 1.7 shows an example of one such scenario, specificallly VF. In this case, the software model matched to the decision of the actual ICD which also determined that therapy should be applied.

In all scenarios, the decision of model conformed to that of the ICD. The remaining two branches were not reachable due to the limited output capability of the programmer. The Medtronic software implementation can be validated using a similar process.

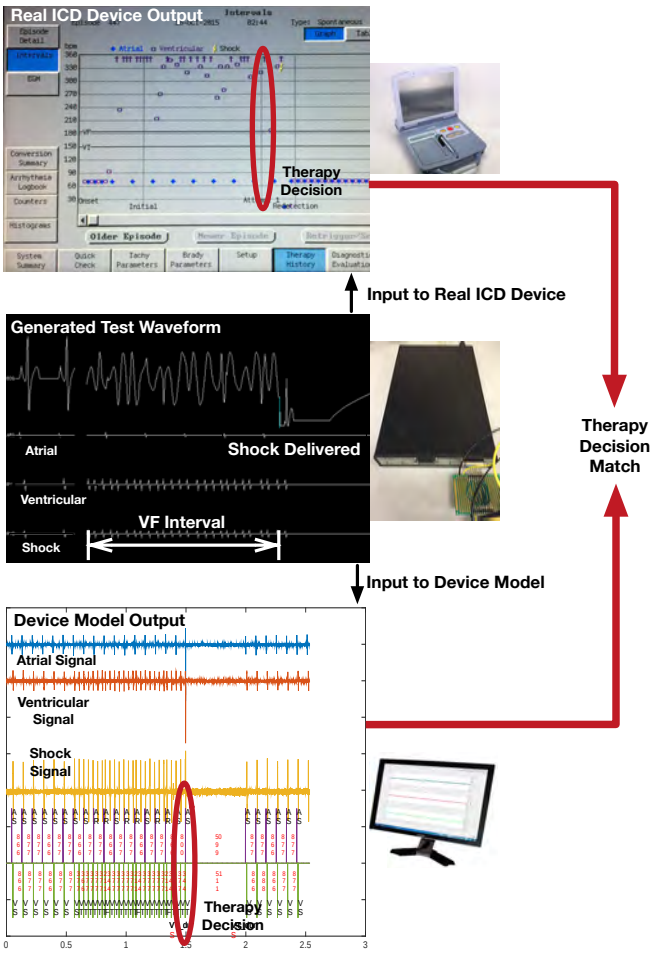


Figure 6.7: Example of validation output screenshots (Ventricular fibrillation) showing matching therapy decision for the ICD and our implementation.

6.4 Results

6.4.1 The rate of inappropriate therapy

The first objective of the in-silico pre-clinical trials is to estimate the rate of inappropriate detection $\bar{t}$ for each of the two algorithms for all arrhythmias combined, i.e., for the entire synthetic cohort. The rate of inappropriate therapy is defined as

$$\bar{t} = \frac{\text{Number of inappropriately applied therapies}}{\text{Number of applied therapies}}$$

From this we can confirm or invalidate the assumption that Rhythm ID outperforms PRL+W. We generated a synthetic cohort of 11,400 heart instances, equally distributed among the 19 arrhythmias. The number of instances was obtained from a Monte Carlo calculation.

Conclusion 1: PRL+W delivers less inappropriate therapy. The obtained rates of inappropriate detection were 6.65% for Boston Scientific and 2.91% for Medtronic (P ¡ 0.0001), assuming an equal number of patients from each arrhythmia in the synthetic cohort. The corresponding relative improvement *of Medtronic over Boston Scientific* is 56%. In other words, the in-silico pre-clinical trial reveals that the PRL+W algorithm from Medtronic actually differentiates between VT and SVT better than Rhythm ID from Boston Scientific. Our findings are consistent with the observations of the RIGHT trial itself (Gold et al. [2012]).

Conclusion 2: result holds across population characteristics. The above rates were obtained under the assumption that each arrhythmia is equally represented in the cohort. A significant feature of in-silico pre-clinical trial is that it allows us to study the endpoint of interest (here, rate of inappropriate detection) on a variety of populations, which have the various arrhythmias in different proportions. This may not be feasible in a real clinical trial, which has to contend with the population present at the clinical centers where the trial is conducted. We may then ask: does PRL+W maintain a lower rate of inappropriate detection across different populations? To answer this question, we varied the distribution of the arrhythmias in the synthetic cohort, and re-computed the cohort-wide rates of inappropriate therapy. Fig. 6.8 shows the results for 10 random variations of the arrhythmia distribution. It can be seen that indeed, PRL+W maintains a better rate of arrhythmia discrimination (and by inference, less inappropriate therapy) across the board.

This illustrates very well the benefit that an in-silico pre-clinical trial can bring to the planning of an RCT: the fact that Rhythm ID could not be shown to be better than PRL+W can cause the investigators to re-consider their assumptions and the feasibility of the trial. In this case, the in-silico pre-clinical trial casts doubt on the assumed *direction* of the effect, i.e. whether intervention is better than control, or the other way around. This early check can mean the difference between an expensive trial that fails at showing the desired effect, and a trial that is appropriately sized to demonstrate the desired effect size.

Thus, while an in-silico pre-clinical trial does not replace or mimic the RCT since we can not capture patient-level outcomes of the therapy, it can provide but *early insight* at a small fraction of the RCT cost and duration and without the ethical issues.

6.4.2 Condition-level rates

Having a heart model allows us to better estimate the *sensitivity* and *specificity* of the diagnostic algorithms' performance, something which is not possible in a clinical trial because the device only records a limited number of episodes. These are defined as

$$\text{Sensitivity} = \frac{\text{Number of correctly classified VTs}}{\text{Number of true VTs}}$$

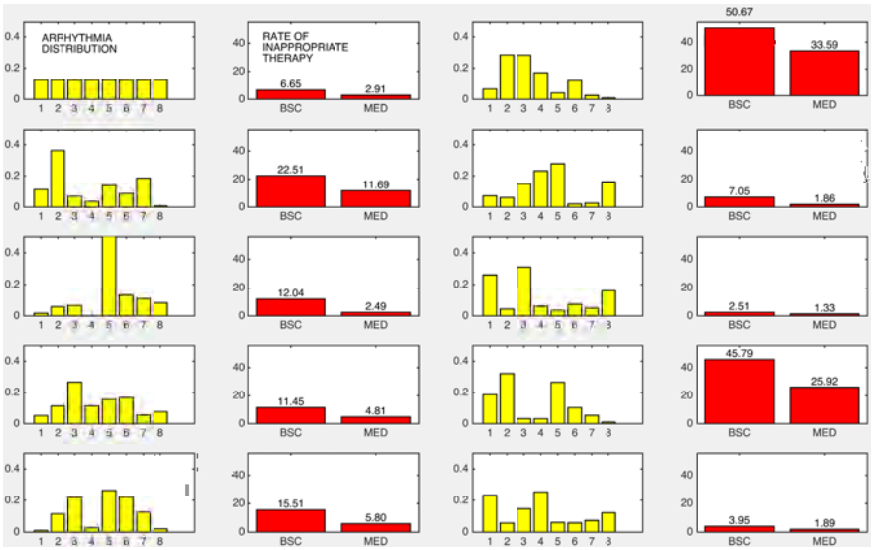


Figure 6.8: Rate of inappropriate detection (2^{nd} and 4^{th} columns) for different arrhythmia distributions (1^{st} and 3^{d} columns). The arrhythmias are (left to right on the x axis): Atrial fibrillation, Atrial flutter, Premature Ventricular Complexes, Nonsustained Ventricular Fibrillation, Supraventricular Tachycardia, Sinus Brady-Tachy, Ventricular Fibrillation, Ventricular Tachycardia (Josephson [2008]). The top left distribution is uniform, and the bottom right distribution is that of the baseline characterization in RIGHT (Gold et al. [2012]).

$$\text{Specificity} = \frac{\text{Number of correctly classified SVTs}}{\text{Number of true SVTs}}$$

In words, the sensitivity measures how well the device recognizes VTs. Specificity measures how well the device discriminates between VT and SVT. An ideal device would have 100% sensitivity and specificity. Unfortunately, these are typically competing goals: the more sensitive the device, the more likely it will mis-diagnose some SVTs as VTs, so its specificity will drop.

We calculated sensitivity and specificity in our in-silico pre-clinical trials, and report them in Table 6.1 on a per-arrhythmia basis. The conditions are drawn from RIGHT's baseline characterization (Gold et al. [2012]). Specificity is reported for SVTs and sensitivity is reported for VTs. It can be seen from these results that in our synthetic cohort, Atrial flutter and other Supraventricular tachycardias are the main source of inappropriate detection for Rhythm ID compared to PRL+W. In the case of Atrial flutter, Rhythm ID categorizes it inappropriately as VT for 41.7% of the cases.

Condition-level analysis pinpoints the specific pathways of the discrimination algorithm which must be addressed to reduce the device's rate of inappropriate therapy. It is difficult to get such insight through an RCT as the patient population is fixed and the conditions are determined retroactively. Such analysis can be further used to investigate condition distributions across different patient population types (e.g. abnormal heart rhythms in children vs geographic region-specific or race-specific con-

Arrhythmia	Rhythm ID	PR Logic + Wavelet	P value
	Specificity (%)		
Atrial Fibrillation	99.8	99.6	0.3167
Atrial flutter	58.3	79.33	< 0.0001
Premature ventricular complexes	100	100	1
Nonsustained ventricular tachycardia	100	99.8	0.3171
Other Supraventricular tachycardia	96.3	99.7	< 0.0001
Brady-Tachy	100	98.83	0.0079
	Sensitivity (%)		**P value**
Ventricular fibrillation	100	100	1
Ventricular tachycardia	100	100	1

Table 6.1: Specificity and sensitivity under different heart conditions.

dition distributions).

6.4.3 Effect of Device Parameters on Discriminating Capability

ICDs have a number of parameters which can be tuned to accommodate specific patient conditions by the physicians. Currently there are very few clinical results on the effect of tuning parameters and their effect on sensitivity and specificity (Moss et al. [2012]). One of the main causes of VT/SVT mis-classifications is inappropriate parameter settings (Daubert et al. [2008]). In order for the physicians to set appropriate parameters, it is very important to understand how the change of one parameter can affect the discriminating capability of the device. It is costly to experiment this on real patients. With in-silico pre-clinical trial, one can use the same population across multiple devices with different parameter settings at virtually no cost.

In this section, we use in-silico pre-clinical trial to demonstrate the effects of changing two common parameters on SVT/VT discrimination specificity. The first parameter is the ***duration*** of arrhythmia before the ICD makes a therapy decision. For Boston Scientific ICD the value can be set to 1 to 30 seconds. In this experiment

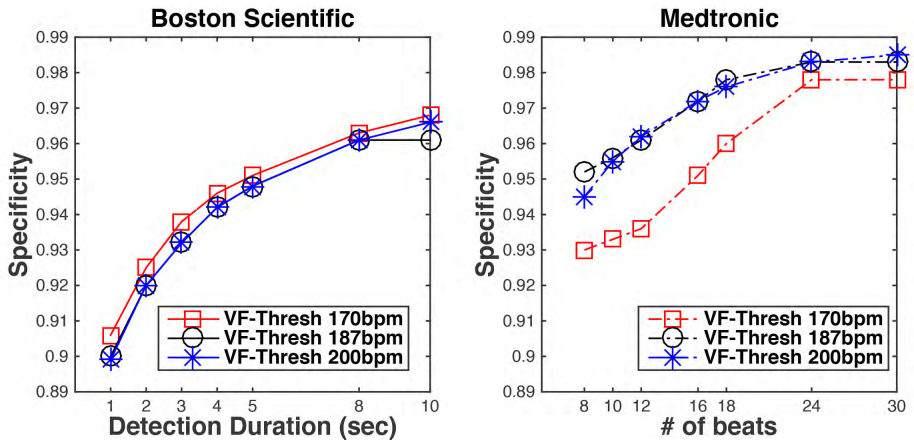


Figure 6.9: Effects of Duration and VF threshold parameters on Specificity

we explore the values {1,2,3,4,5,8,10}. The equivalent parameter for Medtronic ICD is the number of consecutive fast ventricular intervals which can be set from 8 to 20 beats. In this experiment we explore the values {8,10,12,16,18,24,30} which roughly correspond to the parameters of Boston Scientific ICD. Intuitively, with a longer duration the device can examine a longer history of the arrhythmia episode, and also allows a greater chance for the arrhythmia to self-terminate. This can prevent inappropriate therapy. Setting the duration too long can also delay and in some cases withhold appropriate therapy. These results are in agreement with the recently conducted ADVANCE-III RCT which showed that longer arrhythmia detection windows reduce shocks for Medtronic ICDs (Gasparini et al. [2013]).

The second parameter we varied is the ***VF threshold***. For both devices, if the ventricular rate is faster than the VF threshold for a period of time the devices will deliver therapy without going into the SVT/VT discrimination algorithm. So a higher VF threshold means that more signals are passing through the discrimination algorithm. In this experiment we explored the values {170,184,200} for both algorithms. Intuitively the higher the threshold, the more episodes will be examined by the SVT/VT discrimination algorithm, which may increase specificity.

Conducting the Model-based Clinical Trial. For each of the 21 parameter combinations described above, we ran an in-silico pre-clinical trial with 11,400 EGM episodes on both device models. From the results we observe that for both algorithms the specificity increases monotonically with the length of the duration. When the duration is longer than 5, sensitivities dropped below 100%, which is in line with the intuition. However, Rhythm ID and PRL+W displayed opposite trends for the VF threshold. For PRL+W, the specificity increased when the VF threshold was increased from 170BPM to 184BPM - i.e. a higher threshold admits more signals through the discrimination algorithm which performs better across all rates. For Rhythm ID the specificity dropped when the VF threshold was increased from 170BPM to 184BPM - i.e. the discrimination algorithm is less effective at higher rates. One possible interpretation of the result is that the Boston Scientific algorithm

is more prone to inappropriate therapies for SVTs with ventricular rate between 170BPM to 184BPM, which may be a useful for the physicians to consider during parameter settings.

6.5 Discussion

The above experiment has illustrated a practical application of computer modeling for the support of clinical trial planning and execution. We now present the medically relevant limitations of this particular experiment, then discuss the in-silico pre-clinical trial approach in general. First, we did not account for post-shock detection (a phase of detection that follows the delivery of therapy), which was part of the RIGHT results. The Onset discriminator was not implemented so the results exclude its effects. RIGHT included both dual-chamber and single-chamber devices, whereas we only implemented the algorithms for dual-chamber devices. For the VTC and Wavelet algorithms, the literature did not specify which samples are taken from the electrogram. We chose to sample the electrogram uniformly in time, and validated that this gives correct results.

Clinical trials study the effect of an intervention *in the patient*, and report patient-level results (e.g., "The event of interest was observed in X% of patients in Group 1"). Our results are at the condition level: they take the form "the event of interest was observed in X% of generated conditions". To produce patient-level estimates requires an estimate of how conditions are distributed among patients. This low-level data is not readily available.

It is important to stress that in general, one should not expect *absolute numbers* from an in-silico pre-clinical trial to match those from a clinical trial, nor should this be the goal. For example, in this work, it is unlikely that our in-silico pre-clinical trials will yield rates of inappropriate therapy that are equal to the rates obtained by RIGHT itself. The reasons for this are many:

- Many factors that affect the outcomes of the trial (such as changes in patient lifestyle) are not modeled.

- The adjudication of episodes in RIGHT (and other trials) is limited by the fact that only therapy episodes were recorded by the devices. The adjudication process is further limited by the lack of surface EKGs, which makes it hard to reliably distinguish certain atrial arrhythmia. Neither of these is a limitation in in-silico pre-clinical trial since we have the ground truth: we know exactly what arrhythmia is being simulated by the model. Furthermore, the AAEL signals have both device electrograms (EGMs) and the corresponding surface EKGs which allow for precise adjudication.

- Experts may disagree on how to adjudicate the more complex episodes, so our classification of episodes from the AAEL database and the classification of the

RIGHT investigators have an irreducible discrepancy. Again, this will affect the statistics that they and we compute.

That said, we can expect that a good heart model will reveal *the trend* of the results, such as improvement of intervention over control or not, as shown in this paper.

Chapter 7

Conclusion

Closed-loop medical devices like implantable cardiac devices have both diagnostic and therapeutic capabilities and interact with the patient autonomously in closed-loop. Their autonomy makes them among the highest risk devices which require the most stringent regulation. Currently, clinical trials are the primary means to identify risks associated with the closed-loop interaction between the devices and the patients. While such clinical trials are necessary, they are expensive and ineffective for validation of the safety and efficacy of medical device software.

Model-based approaches enable closed-loop evaluation throughout the device life-cycle, which require validated physiological models that represent the closed-loop behaviors of different physiological conditions from the device's perspective. In this effort, implantable cardiac devices are used as example to demonstrate the application of model-based approaches in providing safety and effectiveness towards "regulatory grade evidence" of the device and describe how these activities align into the regulatory process. A heart model structure that captures the electrical behaviors of the heart was developed. The model structure was based on clinical electrophysiology, which can be easily interpreted by physicians and its parameters can be identified from patient data. The model structure was applied in two model-based closed-loop validation applications for implantable cardiac devices.

During device development, the safety and efficacy of a device design can be evaluated using closed-loop model checking. In closed-loop model checking, the variability of patient conditions is captured by applying over-approximating behaviors of the heart models. An abstraction tree framework was developed to capture all possible observable behaviors of a human heart from the device perspective, and provide physiological contexts to the counter-examples returned by the model checker. We demonstrated that closed-loop model checking can effectively identify safety and efficacy violations of device design.

An automated translation tool was developed to translate UPPAAL model to Stateflow chart. The rigorous translation together with automated code generation guarantee that properties validated during model checking also hold in the final device implementation. The complete verified model to verified code toolchain encourages

the application of model-based design for other autonomous medical devices.

Before a device is finally evaluated on real patients in clinical trials, in-silico pre-clinical trials can be performed to further increase the confidence of device safety and efficacy. In in-silico pre-clinical trial, the variability of patient conditions is captured by generating a large number of individual physiological models across a number of physiological conditions. This allows us to explore a wider range of heart behaviors and expose more corner cases to isolate software safety issues prior to an actual clinical trial. in-silico pre-clinical trials for medical device software have the potential to complement the current regulatory approach by reducing the cost, scope and probability of failure of a traditional clinical trials.

The area of Medical Cyber-Physical Systems is in its early days with several exciting and urgent fundamental challenges in modeling, control, verification and testing for higher confidence life-critical systems. Similar approaches used in this dissertation can potentially be applied in developing other autonomous medical devices. The major challenge is the amount of physiological domain expertise required to develop appropriate physiological models. The model developer needs to identify the minimum amount of details required in the model to interact with the device and interpret different physiological conditions. The physiological models are then required to be optimized for appropriate formalism in order to do closed-loop model checking. Variations of the physiological models are also required for different model applications and different physiological requirements. Most importantly, the validity of the physiological models have to be justified to provide confidence to the evaluation results.

Bibliography

Medtronic ViP-II Virtual Interactive Patient: User's Manual Software v1.5. Rivertek Medical Systems, 2006.

R. Alur. *Timed Automata*. Lecture Notes in Computer Science, Springer, 1999.

R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for schedulability analysis and code generation of real-time systems. In *Formal Modeling and Analysis of Timed Systems*, volume 2791, pages 60–72. Springer, 2004.

Avicenna Consortium. In-silico Clinical Trials: How Computer Simulations will Transform the Biomedical Industry. *Roadmap document*, 2015.

A. Ayoub, A. Wahba, A. Salem, and M. Sheirah. Code synthesis for timed automata: A comparison using case study. In *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 403–403. Springer, 2010.

B. P. Kovatchev and M. Breton and C. Dalla Man and C. Cobelli. In Silico Preclinical Trials: A Proof of Concept in Closed-Loop Control of Type 1 Diabetes. *Journal of Diabetes Science and Technology*, 3, 2009.

J. Beaumont, D. C. Michaels, M. Delmar, J. Davidenko, and J. Jalife. A Model Study of Changes in Excitability of Ventricular Muscle Cells. *American Journal of Physiology. 268*, 1995.

G. Behrmann, A. David, and K. G. Larsen. A Tutorial on UPPAAL. *Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science*, pages 200–236, 2004.

J. Bengtsson and W. Yi. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets*, volume 3098, pages 87–124. LNCS, 2004.

R. D Berger et al. The Rhythm ID Going Head to Head Trial (RIGHT): Design of a Randomized Trial Comparing Competitive Rhythm Discrimination Algorithms in Implantable Cardioverter Defibrillators. *Journal of Cardiovascular Electrophysiology*, 17(7):749–753, 2006.

P. Bogdan, S. Jain, and R. Marculescu. Pacemaker control of heart rate variability: A cyber physical system perspective. *ACM Transactions on Embedded Computing Systems*, 12(1s):50:1–50:22, 2013.

Boston Scientific Corporation. PACEMAKER System Specification. Boston Scientific. *Device Documentation*, 2007a.

Boston Scientific Corporation. The Compass - Technical Guide to Boston Scientific Cardiac Rhythm Management Products. *Device Documentation*, 2007b.

C. D. Swerdlow et al . Discrimination of Ventricular Tachycardia from Supraventricular Tachycardia by a Downloaded Wavelet Transform Morphology Algorithm: A Paradigm for Development of Implantable Cardioverter Defibrillator Detection Algorithms. *J. Cardiovascular Electrophysiology*, 13, 2002.

C. Toffanin, M. Messori, F. Di Palma, G. De Nicolao, C. Cobelli, L. Magni. Artificial Pancreas: Model Predictive Control Design from Clinical Experience. *J Diabetes Sci Technol*, 7:1470–1483, 2014.

Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. Quantitative verification of implantable cardiac pacemakers over hybrid heart models. *Information and Computation*, 236(0):87 – 101, 2014.

D. Clarke and I. Lee. Testing real-time constraints in a process algebraic setting. In *Proceedings of the International Conference on Software Engineering*, pages 51–60, 1995.

E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counter Example-Guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM*, 50(5): 752–794, 2003.

E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, 1982.

R J. Coffey. Deep brain stimulation devices: A brief technical history and review. *Artificial Organs*, 33(3):208–220, 2009.

J. M. Cortner. Testing Implantable Medical Devices. *Global Healthcare Medical Device Manufacturing Technology*, pages 2–4, 2003.

Macedo H. D., Larsen P. G., and Fitzgerald J. Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System using VDM. *Formal Methods*, pages 28–30, 2008.

D A. Vogel. *Medical Devices Software: Verification, Validation and Compliance*. Artech House, 2011.

J. P. Daubert et al. Inappropriate Implantable Cardioverter-Defibrillator Shocks in MADIT II: Frequency, Mechanisms, Predictors, and Survival Impact . *Journal of the American College of Cardiology*, 51(14):1357 – 1365, 2008.

K. Ellenbogen, G. N. Kay, C-P Lau, and B. L. Wilkoff. *Clinical Cardiac Pacing, Defibrillation, and Resynchronization Therapy*. Elsevier, 2011.

E.W. Hsu and C.S. Henriquez. Myocardial fiber orientation mapping using reduced-encoding diffusion tensor imaging. *Journal of Cardiovascular Magnetic Resonance*, 3(4):339–347, 2011.

P. Feiler, L. Wrage, and J. Hansson. System architecture virtual integration: A case study. *Embedded Real-time Software and Systems Conference*, 2010.

B. Fuertes and J. Toquero. Pacemaker Lead Displacement: Mechanisms And Management. *Indian Pacing Electrophysiology Journal*, 2003.

S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange. Autosar–a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles*, volume 62, 2009.

M Gasparini et al. Effect of Long-detection Interval vs Standard-detection Interval for Implantable Cardioverter-Defibrillators on Antitachycardia Pacing and Shock Delivery: The ADVANCE III Randomized Clinical Trial. *JAMA*, 309(18):1903–1911, 2013. doi: 10.1001/jama.2013.4598. URL `+http://dx.doi.org/10.1001/jama.2013.4598`.

M. Ghorbani and P. Bogdan. A cyber-physical system approach to artificial pancreas design. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, 2013.

M. Gold et al. Advanced Rhythm Discrimination for Implantable Cardioverter Defibrillators Using Electrogram Vector Timing and Correlation. *J Cardiovasc Electrophysiol*, 2002.

M. R. Gold et al. Prospective comparison of discrimination algorithms to prevent inappropriate ICD therapy: Primary results of the Rhythm ID Going Head to Head Trial . *Heart Rhythm*, 9(3):370 – 377, 2012.

A. O. Gomes and M. V. Oliveira. Formal Specification of a Cardiac Pacing System. *In Proceedings of the 2nd World Congress on Formal Methods (FM '09)*, pages 692–707, 2009a.

A. O. Gomes and M. V. Oliveira. Formal Specification of a Cardiac Pacing System. In *Proceedings of the 2nd World Congress on Formal Methods*, FM '09, pages 692–707. Springer-Verlag, 2009b.

R. Grosu, G. Batt, F. H. Fenton, J. Glimm, C. Le Guernic, S.A. Smolka, and E. Bartocci. From cardiac cells to genetic regulatory networks. In *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 396–411. Springer Berlin Heidelberg, 2011.

G. Hamon. A Denotational Semantics for Stateflow. In *EMSOFT'05: Proc. of the 5th ACM international conference on Embedded software*, pages 164–172, 2005.

G. Hamon and J. Rushby. An Operational Semantics for Stateflow. *International Journal on Software Tools for Technology Transfer*, 9(5):447–456, 2007.

R. G. Hauser and B. J. Maron. Lessons from the Failure and Recall of an Implantable Cardioverter-Defibrillator. *"American Heart Association, Circulation"*, pages 2040–2042, 2005.

M. Hendriks. Translating UPPAAL to Not Quite C. Technical Report CSI-R0108, Comp. Sc. Institute, 2001.

http://revisionworld.com/.

Ask The ICD. `http://asktheicd.com`, 2015. Accessed on 10/11/2015.

Mathworks Inc. Matlab R2011a Stateflow Documentation. http://www.mathworks.com/help/toolbox/stateflow, 2016.

Md. A. Islam, A. Murthy, A. Girard, S. A. Smolka, and R. Grosu. Compositionality results for cardiac cell dynamics. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, (HSCC '14), pages 243–252, 2014.

E. Jee, I. Lee, and O. Sokolsky. Assurance Cases in Model-Driven Development of the Pacemaker Software. In *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *LNCS*, pages 343–356. 2010a.

E. Jee, S. Wang, J. K. Kim, J. Lee, O. Sokolsky, and I. Lee. A Safety-Assured Development Approach for Real-Time Software. *The Proceedings of 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 133–142, 2010b.

J. M Jenkins and R. E Jenkins. Arrhythmia database for algorithm testing: surface leads plus intracardiac leads for validation. *Journal of Electrocardiology*, 36, Supplement 1:157 – 161, 2003.

R. Jetley, S. P. Iyer, and P. L. Jones. A Formal Methods Approach to Medical Device Review. *IEEE Computer*, 39:61–67, 2006.

Z. Jiang and R. Mangharam. Modeling Cardiac Pacemaker Malfunctions with the Virtual Heart Model. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 263 –266, Sept 2011.

Z. Jiang, A. Connolly, and R. Mangharam. Using the Virtual Heart Model to Validate the Mode-Switch Pacemaker Operation. *IEEE Engineering in Medicine and Biology Society*, pages 6690 –6693, 2010a.

Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam. Real-time heart model for implantable cardiac device validation and verification. In *2010 22nd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 239 –248, July 2010b.

Z. Jiang, M. Pajic, and R. Mangharam. Model-based Closed-loop Testing of Implantable Pacemakers. In *ACM/IEEE Second International Conference on Cyber-Physical Systems (ICCP'11)*, 2011.

Z. Jiang, M. Pajic, and R. Mangharam. Cyber-Physical Modeling of Implantable Cardiac Medical Devices. *Proceedings of the IEEE*, 100(1):122 –137, Jan. 2012a.

Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and Verification of a Dual Chamber Implantable Pacemaker. *Tools and Algorithms for the Construction and Analysis of Systems*, 7214:188–203, 2012b.

Z. Jiang, M. Pajic, R. Alur, and R. Mangharam. Closed-loop verification of medical devices with model abstraction and refinement. *International Journal on Software Tools for Technology Transfer*, 16(2):191–213, 2014.

Z. Jiang, H. Abbas, PJ. Mosterman, and R. Mangharam. Tech Report: Abstraction-Tree For Closed-loop Model Checking of Medical Devices. *http://repository.upenn.edu/mlab_papers/73*, 2015.

M. E. Josephson. *Clinical Cardiac Electrophysiology*. Lippincot Williams and Wilkins, 2008.

A. V. Kaplan et al. Medical device development: From prototype to regulatory approval. *Circulation*, 109(25):3068–3072, 2004. doi: 10.1161/01.CIR.0000134695.65733.64. URL `http://circ.ahajournals.org/content/109/25/3068.short`.

B. G. Kim, A. Ayoub, P. Jones, O. Sokolsky Y. Zhang, R. Jetley, and I. Lee. Safety-assured development of the gpca infusion pump software. In *EMSOFT'11: ACM conf. on Embedded software*, pages 89–98, 2011.

L. M. Friedman and C. D. Furberg and D. L. DeMets. *Fundamentals of clinical trials*. Springer, 2010.

K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, pages 134–152, 1997.

W. H. Maisel, M. O. Sweeney, W. G. Stevenson, K. E. Ellison, and L. M. Epstein. Recalls and Safety Alerts involving Pacemakers and Implantable Cardioverter-Defibrillator Generators. *JAMA*, 286(7), 2001.

D. Mery and N. K. Singh. Pacemaker's Functional Behaviors in Event-B. *Research report, INRIA*, 2009.

A. J. Moss et al. Reduction in inappropriate therapy and mortality through icd programming. *New England Journal of Medicine*, 367(24):2275–2283, 2012. doi: 10.1056/NEJMoa1211107.

A. Murthy, E. Bartocci, F. H. Fenton, J. Glimm, R. A. Gray, E. M. Cherry, S. A. Smolka, and R. Grosu. Curvature analysis of cardiac excitation wavefronts. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(2): 323–336, 2013.

Nano-RK. Nano-RK Sensor RTOS, Carnegie Mellon University. *http://nanork.org*, 2007.

M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In *18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 173–184, 2012a.

M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From Verification to Implementation: A Model Translation Tool and a Pacemaker Case Study. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, RTAS '12, pages 173–184, 2012b.

M. Pajic, I. Lee, R. Mangharam, and O. Sokolsky. UPP2SF: Translating UPPAAL models to Simulink. Technical report, University of Pennsylvania, 2012c.

C. S. Peskin and D. M. McQueen. A three-dimensional computational method for blood flow in the heart. 1. immersed elastic fibers in a viscous incompressible fluid. *Journal of Computer Physics*, 81(2):372–405, 1989.

Research and Markets. Global medical devices market 2012-2020: Market size, share, trends, analysis and forecast. *Grace Market Data*, 2015.

M. Rosenqvist, T. Beyer, M. Block, K. Dulk, J. Minten, and F. Lindemans. Adverse Events with Transvenous Implantable Cardioverter-Defibrillators: A Prospective Multi-center Study. *Circulation*, 1998.

S. Rossi, R. Ruiz-Baier, L. F. Pavarino, and A. Quarteroni. Active strain and activation models in cardiac electromechanics. *Proceedings in Applied Mathematics and Mechanics (PAMM)*, 11(1):119–120, 2011.

F. B. Sachse, A. P. Moreno, and J. A. Abildskov. Electrophysiological modeling of fibroblasts and their interaction with myocytes. *Annals of Biomedical Engineering*, 36(1):41–56, 2008.

K. Sandler, L. Ohrstrom, L. Moy, and R. McVay. Killed by Code: Software Transparency in Implantable Medical Devices. *Software Freedom Law Center*, 2010.

S. J. Saxonhouse, J. B. Conti, and A. B. Curis. Current of injury predicts adequate active lead fixation in permanent pacemaker/defibrillation leads. *Journal of the American college of Cardiology*, 2005.

N. Scaife, C. Sofronis, P. Caspi, S. Tripakis, and F. Maraninchi. Defining and Translating a "Safe" Subset of Simulink/Stateflow into Lustre. In *EMSOFT'04: ACM conf. on Embedded software*, pages 259–268, 2004.

R. Schulte, G. Sands, F. Sachse, O. Dossel, and A. Pullan. Creation of a Human Heart, Model and its Customisation using Ultrasound Images. *Biomedizinische Technik/Biomedical Engineering*, 46:26–28, 2001.

Igor Singer. *Interventional Electrophysiology*. Lippincott William & Wilkins, 2001.

W. Stevenson and K. Soejima. Recording Techniques for Clinical Electrophysiology. *Journal of Cardiovascular Electrophysiology*, 16:1017–1022, 2005.

Kirsten H.W.J. Ten Tusscher, Rok Hren, and Alexander V. Panfilov. Organization of ventricular fibrillation in the human heart. *Circulation Research*, 100(12): e87–e101, 2007. doi: 10.1161/CIRCRESAHA.107.150730. URL `http://circres.ahajournals.org/content/100/12/e87.abstract`.

N. A. Trayanova and P. M. Boyle. Advances in modeling ventricular arrhythmias: from mechanisms to the clinic. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 6(2):209–224, 2014.

L. A. Tuan, M. C. Zheng, and Q. T. Tho. Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker. *Fourth International Conference on Secure Software Integration and Reliability Improvement*, pages 23–32, 2010.

U.S.FDA. Design Control Guidance For Medical Device Manufacturers. *Center for Devices and Radiological Health*, 1997.

U.S.FDA. General principles of software validation; final guidance for industry and fda staff. *Center for Devices and Radiological Health*, 2002.

U.S.FDA. Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices. *Center for Devices and Radiological Health*, 2005.

U.S.FDA. Ensuring the Safety of Marketed Medical Devices: CDRH's Medical Device Postmarket Safety Program. *Center for Devices and Radiological Health*, Jan 2006.

U.S.FDA. Medical device recall report - fy2003 to fy2012. *Center for Devices and Radiological Health*, 2012.

U.S.FDA. Human Subject Protection; Acceptance of Data from Clinical Studies for Medical Devices; Proposed Rule. *Docket No. FDA- 2013-N-0080*, 2013.

U.S.FDA. Classification of medical devices. *US FDA documents*, 2014.

J. E. Wiggelinkhuizen. Feasibility of Formal Model Checking in the Vitatron Environment. *Master thesis, Eindhoven University of Technology*, 2007.

S. Yamane. Timed Weak Simulation Verification and its Application to Stepwise Refinement of Real Time Software. *International Journal of Computer Science and Network Security*, 6, 2006.

S. Zhang, C. Kriza, S. Schaller, and P. L. Kolominsky-Rabas. Recalls of cardiac implants in the last decade: what lessons can we learn? *PLoS ONE 10(5): e0125987.*, 2015. doi: doi:10.1371/journal.pone.0125987.