Analyzing Requirements Using Environment Modelling

Dominique Méry[1] and Neeraj Kumar Singh[2(✉)]

[1] Université de Lorraine, LORIA, BP 239, Nancy, France
Dominique.Mery@loria.fr
[2] McMaster Centre for Software Certification,
McMaster University, Hamilton, Canada
singhn10@mcmaster.ca

Abstract. Analysing requirements is a major challenge in the area of safety-critical software, where requirements quality is an important issue to build a dependable critical system. Most of the time, any project fails due to lack of understanding of *user needs*, missing functional and non-functional system requirements, inadequate methods and tools, and inconsistent system specification. This often results from the poor quality of system requirements. Based on our experience and knowledge, an environment model has been recognized to be a promising approach to support requirements engineering to validate a system specification. It is crucial to get an approval and feedback in early stage of system development to ensure completeness and correctness of requirements specification. In this paper, we propose a method for analysing system requirements using a closed-loop modelling technique. A closed-loop model is an integration of system model and environment model, where both the system and environment models are formalized using formal techniques. Formal verification of the closed-loop model helps to identify missing system requirements or new emergent behaviours, which are not covered earlier during the requirements elicitation process. Moreover, an environment model assists in the construction, clarification, and validation of the given system requirements.

Keywords: Environment modelling · Closed-loop modelling · Analysing requirements · Verification

1 Introduction

Requirements engineering (RE) provides a framework for simplifying a complex system to get a better understanding of system requirements by using several formal and informal techniques. It plays an important role in early stage of system development to meet system qualities, success of the system, and reducing the cost of overall development. The prime causes of project failure are lack of understanding of system behaviour, missing functional and non-functional requirements, and inconsistent system requirements, which often lead to poor quality of requirements specification. Increasing complexities and system requirements require to

V.G. Duffy (Ed.): DHM 2015, Part II, LNCS 9185, pp. 345–357, 2015.
DOI: 10.1007/978-3-319-21070-4_35

pay more attention towards requirements engineering to omit a system failure [1]. "*what the system should do?*" is an initial goal of requirements engineering. Incompleteness, ambiguity, inconsistencies, and vagueness, are the most common problems encountered during the elicitation and specification of system requirements. However, the prime objective of any requirements engineering tool is to address these common problems [1].

To identify missing system requirements, inconsistency or new emergent behaviours in early stage of system development for developing a quality, safety, and dependable system, we need to look beyond the system itself, and into the working environment, including human interactions. Requirements traceability is a branch of the requirements management within the software development. Requirements traceability is concerned with documenting the life of requirements and to allow bi-directional traceability in the system requirements, and product produced like source codes and test cases. It enables users to identify the origin of each requirement and track every change, which was made to this requirement. Validation of requirements specification is an integral and essential part of requirements engineering. Validation is a process of checking, together with stakeholders, whether the requirements specification meets its stakeholders' intentions and expectations [2].

In this paper, we propose a method for analysing system requirements using environment modelling. The environment model and system model form a closed-loop system to trace missing requirements or new emergent behaviours. The closed-loop model is an integration of system model and environment model, where both the system and environment models are formalized using formal techniques. Formal verification of this closed-loop model helps to identify missing system requirements or new emergent behaviours, which are not covered earlier during the requirements elicitation process. This closed-loop modelling approach helps to get confidence in early stage of system development. This approach offers numerous benefits of the proposed approach as follows:

- Closed-loop modelling in early stage of system development;
- Identifying gaps or inconsistencies in system requirements;
- Strengthening the given system requirements;
- To support "what-if" analysis during formal reasoning;
- Traceability of missing behaviours that leave the system in undesired state;
- Automatic identification of an emergent behaviour;
- Validation of the system assumptions;
- Financial benefits that can allow to change the system requirements.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 presents a short description of requirements engineering. Section 4 presents an overview of environment modelling. Section 5 presents a case study to model a closed-loop system for verifying system requirements. Section 6 discusses the environment modelling approach for analysing system requirements. Section 7 concludes the paper.

2 Related Work

Requirements analysis is an important step in the software development lifecycle that allows for eliciting, analysing, and recording the system requirements. The requirements should be unambiguous, satisfying completeness, formally verified, proper documented, and traceable. A list of errors related to software requirements is given in [3]. This paper also presents a detailed survey on different types of errors that are the root causes for failing the critical systems. The root causes are program faults, human errors, and process flaws. There are several methods are applicable for requirements analysis like prototyping and simulation. The exiting methodologies are not sufficient, and we are still striving for better methods for improving the requirements analysis process.

Prototype refers to an incomplete version of the system development that simulates a system partially when system requirements are indefinite and system behaviours are unclear [4]. Goguen et al. [5] have proposed a novel approach for constructing a prototype using formal specification, where this work advocates the use of an algebraic specification language for executing the given specification. A run-time technique for monitoring the system requirements for satisfaction purpose is presented in [6], where the system requirements are monitored for violations, and system behaviour is dynamically adapted a new behaviour. New introduced behaviour changes the system requirements that must meet the higher-level goals.

There are few approaches reported in the literature related to environment modelling. Kishi et al. [7] have proposed an environment modelling approach for an embedded system. A new language is proposed in [8] for modelling an environment and required behaviour to simulate an environment. The UML class diagram and sequence diagram are also used together for modelling and simulating an environment in [9]. Kreiner et al. [10] have presented a process to develop an environment model for simulation purpose, where this environment model can be used to simulate the automatic logistic systems.

An environment modelling is not limited for simulation purpose only. There are some works reported in the literatures that discuss testing based on an environment of a system. This type of environment modelling is applicable for rigorous testing of a given system. Auguston et al. [11] have discussed the development of environment behavioural models using Attributed Event Grammar for testing an embedded system. Heisel et al. [12] have proposed the use of a requirement model and an environment model using the UML state machines for testing. A testing approach for synchronous reactive software is presented in [13] using the temporal logic based on the environmental constraints.

3 Requirements Engineering

Requirements characterize a system related to functional and non-functional behaviours, system properties and safety constraints that must be satisfied by a developing system. The Institute of Electrical and Electronics Engineers (IEEE)

defines a requirement as a condition or capability that must be met or possessed by a system or system components to satisfy the contract, standard, specification, or other formally imposed document [14]. Requirements engineering allows the use of systematic techniques for ensuring completeness, consistency and relevance of system requirements [15]. Requirements engineering has numerous phases for requirements elicitation, analysis, specification, verification, and management. The requirement elicitation is a process for identifying, reviewing, checking, and documenting a stakeholder needs for a given system. The requirements analysis is a process that allows to check a stakeholder needs and system constraints using formal and informal techniques. The requirements specification is a process for documenting a stakeholder needs and constraints unambiguously and precisely using formal or semi-formal techniques. The requirements verification allows to ensure completeness, correctness, understandable and consistent of system behaviour according to stakeholders. The requirements management uses for managing, coordinating, and documenting the system development life-cycle.

In requirements engineering the elicitation process is important to capture rationales and sources precisely in order to understand the requirements evolution and verification. Requirements analysis advocates desirable properties of the software development processes, and numerous problems are identified during the development process. It involves for finding variations and commonalities in the system development process. Moreover, it also allows feedback mechanism to provide essential information for reducing complexity by eliminating complex requirements by simple requirements. There are several techniques that are useful for improving the quality of requirements for dependable critical systems. In this paper, we propose a new technique for analysing requirements using environment modelling by developing a closed-loop model. A closed-loop model is an integration of system model and environment model, where both the system and environment models are specified using formal modelling language. This approach has potential benefits to trace missing requirements or new emergent behaviours. All these approaches, methods, techniques and tools proposed for analysing the requirements are useful as long as its adoption decision is present preferably during the early stages of the projects, and we need to understand how a decision on analysing requirements is made and which factors influence an adoption of the requirements engineering. Here, we present the conceptual treatment for analysing the system requirements using environment modelling for identifying peculiar requirements, which eventually provide us with a theoretical lens to examine this adoption in a systematic manner.

4 Environment Modelling

If environment models are to be used for safe dependable critical systems, they should not only be sufficiently detailed, but should also be easy to understand and modify as an environment and critical system evolve. To handle the complexity of a realistic system environment, a modelling language should have provision for modelling at several levels of abstraction. A modelling language should also

have well-defined syntax and semantics for the tools to specify an environment model accurately that should be understandable by humans. A modelling language should also provide features for modelling real world concepts, real-time features, and other concepts, such as non-determinism, required by the components of an environment. Formal methods based modelling languages, such as Event-B, Z and VDM can be used to fulfill the required features for modelling an environment.

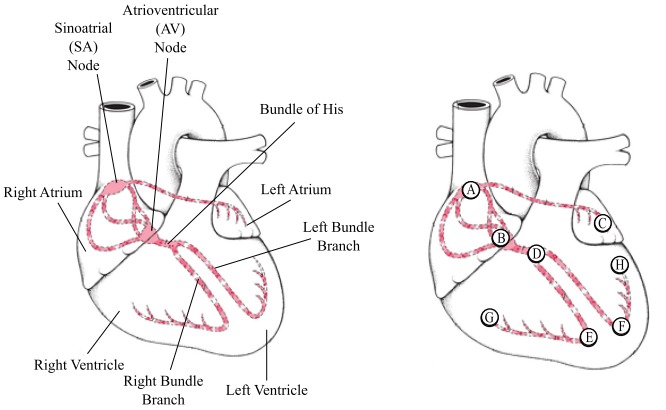
In this work, we are using the same notations to model an environment that are used for modelling the software systems. It is important to note that the methodology for environment modelling is significantly different from the system modelling. While modelling for an industrial case, we have abstracted the functional details of the environment components to an extent that hide the system complexities. For modelling an environment behaviour, non-determinism is widely used, which is not nearly as common when modelling an internal behaviour of a system.

For verifying and tracing the missing requirements or new emergent behaviours of a system based on its environment, the behaviour details of the environment are as important as its structural details. Structural details of a critical system environment are important to understand the overall composition of an environment using actuators and sensors, characteristics of various components, and their relationships. We select the Event-B modelling language to model these details using stepwise incremental refinement, where each refinement step is built by the *correct-by-construction* approach. The incremental refinement also adds the required safety properties to develop a safe and correct environment. The behavioural details of environment components are required to specify the dynamic aspects of an environment, for example, to determine the possible environment states, before and after its interactions whenever system is in effect, and to specify the possible interactions between the system and its environment.

In the following subsections, we discuss a modelling methodology for developing an environment model (the heart). Then the heart model will be used to develop a closed-loop model for a cardiac pacemaker. This closed-loop model of the pacemaker and heart can be used further for analysing system requirements and to identify missing requirements or new emergent behaviours.

4.1 Heart Model

The heart is a muscular organ that functions as the body's circulatory pump. It contains four chamber (see Fig. 1(a)), which contract and relax periodically. For contracting and relaxing, the heart requires an electrical stimulus that is generated by the sinus node. This electrical stimulus travels down through the conduction pathways. The electrical current flows progressively in the heart muscle using special conduction cells. To design an environment model of the heart, we select a set of landmark nodes (A, B, C, D, E, F, G, H) on the conduction network (see Fig. 1(b)), that controls the contraction and relaxing functionalities of the heart. Here, we describe only core idea for developing the heart model. Interested readers can find the modelling process in [16,17]. We introduce the necessary elements using formal notations to define the heart system as follows:



(a) Electrical Conduction system (b) Landmarks in network

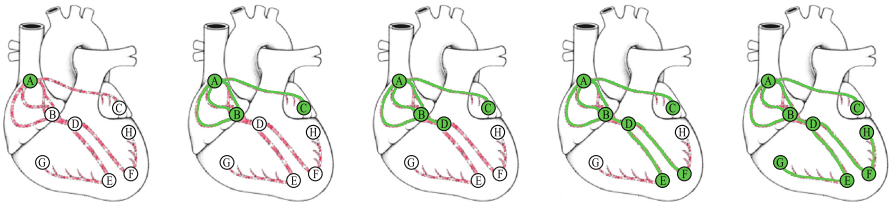
Fig. 1. The electrical conduction and landmarks of the heart system [16]

Definition 1 (The Heart System). *Given a set of nodes N, a transition (conduction) t is a pair (i, j), with i, j ∈ N. A transition is denoted by i ⇝ j. The heart system is a tuple HSys = (N, T, N_0, TW_{time}, CW_{speed}) where:*

- $N = \{ A, B, C, D, E, F, G, H \}$ *is a finite set of landmark nodes in the conduction pathways of the heart system;*
- $T \subseteq N \times N = \{ A \mapsto B, A \mapsto C, B \mapsto D, D \mapsto E, D \mapsto F, E \mapsto G, F \mapsto H \}$ *is a set of transitions to represent electrical impulse propagation between two landmark nodes;*
- $N_0 = A$ *is the initial landmark node (SA node);*
- $TW_{time} \in N \rightarrow TIME$ *is a weight function as time delay of each node, where* $TIME$ *is a range of time delays;*
- $CW_{speed} \in T \rightarrow SPEED$ *is a weight function for the impulse propagation speed of each transition, where* $SPEED$ *is a range of propagation speeds.*

Property 1 (Impulse Propagation Time). *In the heart system, the electrical impulse originates from the SA node (node A), travels through the entire conduction network and terminates at the atrial muscle fibres (node C) and at the ends of the Purkinje fibres in both sides of the ventricular chambers (node G and node H). The impulse propagation time delay differs for each landmark node. The impulse propagation time is represented as the total function* $TW_{time} \in N \rightarrow \mathbb{P}(0..230)$. *The impulse propagation time delay for each node is represented as:* $TW_{time}(A) = 0..10$, $TW_{time}(B) = 50..70$, $TW_{time}(C) = 70..90$, $TW_{time}(D) = 125..160$, $TW_{time}(E) = 145..180$, $TW_{time}(F) = 145..180$, $TW_{time}(G) = 150..210$ *and* $TW_{time}(h) = 150..230$.

Property 2 (Impulse Propagation Speed). *The impulse propagation speed also differs for each transition (i ⇝ j, where i, j ∈ N). The impulse propagation speed is represented as the total function* $CW_{speed} \in T \rightarrow \mathbb{P}(5..400)$. *The Impulse propagation speed for each transition is represented as:* $CW_{speed}(A \mapsto B) =$



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e) Step 5

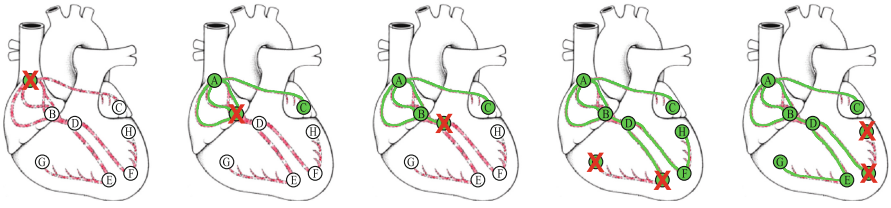
Fig. 2. Impulse propagation through landmark nodes [16]

$30..50$, $CW_{speed}(A \mapsto C) = 30..50$, $CW_{speed}(B \mapsto D) = 100..200$, $CW_{speed}(D \mapsto E) = 100..200$, $CW_{speed}(E \mapsto G) = 300..400$ *and* $CW_{speed}(F \mapsto H) = 300..400$.

The generated electrical stimulus from the sinus node propagates through the conduction network and selected landmark nodes (see Fig. 2). This electrical activity synchronizes the contraction of atria and ventricles. Changing conduction speed affects the natural rhythm and produces abnormalities. The abnormalities lead to various types of arrhythmias. The *bradycardia* is generated due to slow conduction speed, and the *tachycardia* is generated due to fast conduction speed. *Property* 1 and *Property* 2 describe a range of values for impulse propagation time delay for each landmark node and impulse propagation speed for each conduction path, respectively.

The heart blocking is an important term for describing a disorder of conduction of the impulse that stimulates heart muscle contraction. Disturbances in conduction may appear as slow conduction, intermittent conduction failure or complete conduction failure. These three kinds of conduction failure are also known as 1st, 2nd and 3rd degree blocks. We can show these different kinds of heart block throughout the conduction network using selected landmark nodes (see Fig. 3).

A set of spatially distributed cells forms a CA (Cellular Automata) model, which contains a uniform connection pattern among neighbouring cells and local computation laws. The CA is a discrete dynamic system corresponding to space and time that provides uniform properties for state transitions and intercon-



(a) SA Block (b) AV Block (c) Infra-Hisian (d) RBBB (e) LBBB

Fig. 3. Impairments in impulse propagation due to the heart blocks [16]

nection patterns. The cardiac muscle cells of the heart are presented in the following states: *Active*, *Passive* or *Refractory*. Initially, all cells are *Passive*, where each cell is discharged electrically and has no influence on its neighbouring cells. When an electrical impulse propagates, the cell becomes charged and eventually activated (*Active* state). The *Active* cell transmits an electrical impulse to its neighbour cells. The electrical impulse is propagated to all the cells in the heart muscle. After activation, the cell becomes discharged and enters in the *Refractory* state within which the cell cannot be reactivated. After a time, the cell changes its state to the *Passive* state to await the next impulse [17].

5 Case Studies

This section describes a closed-loop model of the cardiac pacemaker and heart, where the cardiac pacemaker responses according to functional behaviour of the heart [16,17]. The main objective of this closed-loop model is for finding inconsistencies, verifying essential safety properties, identifying an emergent behaviour, and strengthening the given system requirements. Due to limited space, we only provide a brief description of the closed-loop system development. A detailed formalization, including safety properties is available in [17,18].

5.1 Abstract Model

To define an abstract model of a closed-loop system, we develop a combined model of the cardiac pacemaker and heart, where the cardiac pacemaker actuates and senses according to the heart behaviour. The environment model of the heart behaves according to observations of an impulse propagation in the conduction network. For developing an abstract model, initially we capture the electrical features thorough defining a set of landmark nodes, impulse propagation times, impulse propagation paths, and impulse propagation speeds. The given parameters describe possible normal and abnormal behaviours of the heart. The cardiac pacemaker model contains sensors, actuators, and timing intervals. The timing intervals are upper rate limit, lower rate limit, and refectory periods for atria and ventricular chambers. The abstract model contains numerous events related to the heart and cardiac pacemaker models to describe the closed-loop model considering several interesting safety properties to establish a desired behaviour for satisfying the given system requirements. We introduce a clock, where time increases progressively by 1 (ms). This clock event controls the time line of pacing and sensing activities including heart impulse propagation.

5.2 Refinement 1: Threshold and Impulse Propagation

This refinement step introduces the impulse propagation that originates from the sinus node and travels down through the conduction network using the landmark nodes. The electrical impulse reaches at the Purkinje fibers of the ventricles. We describe the impulse propagation activities using a set of events, where electrical

impulse passes through the several intermediate landmark nodes and finally sinks to the terminal nodes (C, G, H). The conduction model uses a clock counter to model a real-time system to satisfy the required temporal properties. A set of new events simulates a desired behaviour of the impulse propagation into the heart conduction network, where each new refined event formalizes impulse flow between two landmark nodes. The cardiac pacemaker model is enriched by introducing concrete behaviour of sensors for both the atrial and ventricular chambers, where sensors filter a desired sensing value through comparing with selected standard threshold values. The standard threshold value for atria is always lower than the ventricular chamber. The heart conduction behaviour is continue monitored by the cardiac pacemaker model to allow or inhibit to pace into the heart chamber to control a desired behaviour of the heart according to the standard threshold value under the required timing intervals.

5.3 Refinement 2: Hysteresis and Perturbation the Conduction

This refinement step introduces an abnormal behaviour by introducing the heart blocking activities, and *hysteresis* operating mode. The blocking behaviour specifies perturbation in the heart conduction network to realize an actual abnormal behaviour of the heart. A set of events is introduced using progressive refinement to simulate possible desired blocking behaviours. This blocking activities generate abnormalities into the electrical impulse propagation, which are destined through partition the landmark nodes in the conduction network. The cardiac pacemaker model introduces a new operating mode known as *hysteresis*, which prevents the constant pacing. This mode allows a patient to have his or her own underlying rhythm as much as possible. The *hysteresis* is a programmed feature whereby the pacemaker paces at a faster rate than the sensing rate. A list of events describes this feature for the cardiac pacemaker model.

5.4 Refinement 3: Rate Modulation and Cellular Model

This is the last refinement of the closed-loop system, where a cellular level description is added to the heart, and the rate modulation is added to the cardiac pacemaker. This refinement provides a simulation model that allows to describe the impulse propagation at the cellular level using cellular automata. An electrical impulse propagates at the cells level. A set of events is used to formalize a desired behaviour of the heart using cellular automata. The cardiac pacemaker model is enriched by describing a rate adapting pacing technique. The rate adapting pacing technique gives freedom to select automatically a desired pacing rate according to physiological needs. Automatic selection of a desired pacing rate helps to increase or decrease the pacing rate and assists a patient for controlling the heart rate according to daily activities. The rate modulation sensor is used to determine the maximum exertion performed by a patient. This increased pacing rate refers to the *sensor indicated rate*. Reducing the physical activities helps to progressively decrease the pacing rate down to the lower rate. A set of new refined events models increasing and decreasing pacing rates of the cardiac pacemaker.

5.5 Proof Statics

In this section, we briefly discuss how a closed-loop system modelling approach can be used to analyse system requirements for finding missing requirements. Table 1 expresses the proof statistics of the development of the closed-loop model of the cardiac pacemaker within the heart environment. These statistics measure the size of the model, the proof obligations (POs) generated and discharged by the Rodin prover and those that are interactively proved.

Table 1. Proof statistics

Model	Total number of POs	Automatic proof	Interactive proof
Closed-loop model of One-electrode pacemaker			
Abstract model	304	258 (85 %)	46 (15 %)
First refinement	1015	730 (72 %)	285 (28 %)
Second refinement	72	8 (11 %)	64 (89 %)
Third refinement	153	79 (52 %)	74 (48 %)
Closed-loop model of Two-electrode pacemaker			
Abstract model	291	244 (84 %)	47 (16 %)
First refinement	1039	766 (74 %)	273 (26 %)
Second refinement	53	2 (4 %)	51 (96 %)
Third refinement	122	60 (49 %)	62 (51 %)
Total	3049	2147 (70 %)	902 (30 %)

This closed-loop development results in 3049 (100 %) POs, in which 2147 (70 %) POs are proved automatically, and the remaining 902 (30 %) POs are proved interactively using the Rodin prover. The environment (heart) model is used together with the system (pacemaker) model for verification purpose, that generates automated oracles. These oracles are new proof obligations those are not appeared independently in the system (pacemaker) and environment (heart) models. These generated proof obligations are produced according to the expected system behaviour corresponding to the environment model. A set of new generated POs helps to discover new behaviours by checking formal proofs or undischarged POs. However this approach also helps to check validation of the given assumptions, and identification of new emergent behaviours in the cardiac pacemaker.

6 Analysing Requirements Based on Environment Models

In this section, we briefly discuss how our environment modelling approach is used to analyse system requirements for identifying missing system requirements or new emergent behaviours. The formal model of environment (heart) in Event-B describes the environmental properties in various refinement layers. The developed environment model is used together with the system (cardiac pacemaker) model for verification purpose, that generates automated oracles. These oracles are new proof obligations that do not appear independently in the system model and in the environment model. These generated POs are produced according to the expected system behaviour corresponding to the environment model. However, these generated POs also allow to have more precise actions corresponding to the given guards that should be strengthen to meet expected dynamic behaviours of the cardiac pacemaker, including timing constraints and

pacing and sensing activities. Moreover, the closed-loop model can be used for various purposes during the system development, such as automated code generation and automated test case generation.

In this work, we have focused on formalizing the closed-loop system behaviour of a cardiac pacemaker using several incremental refinements. The goal of this closed-loop model is to provide the nondeterministic behaviour such that an environmental error state is reached during the formal verification, if any fault is present. In fact, to have effective heuristics we need to have precise knowledge of the error states. This information is easily added in the models using stereotypes. All the relevant states/transitions that lead to those error states can be exploited for the automatic derivation of a desired function.

In some relevant cases, it is possible to automatically derive very precise desired functions. This happens when time constraints need to be satisfied, for example a cardiac pacemaker must actuate at exact time interval. Modellers do not need to write these heuristics, they are in fact automatically derived from the given environment model. This is essential, because in general software modellers do not have access to such expertise to write proper desired functions for search algorithms. The results of this experiments show that our closed-loop modelling methodology can be used for a fully automated system verification that is effective in revealing new emergent behaviours, missing system requirements and inconsistencies in the given system. Although the system modelling and environment modelling can be designed in many ways using different strategies, the closed-loop modelling methodology and analysing requirements technique described here would still remain the same.

7 Conclusion

In this paper, we have discussed system requirements analysis using environment modelling. The integration of system model and environment model forms a closed-loop system to trace new emergent behaviours or missing requirements. For practical reason and to facilitate to handle a large complex system, we use the Event-B modelling language to support incremental refinement for modelling, structuring and defining the safety constraints. We have briefly discussed how an environment model is used to simulate required operating environment for a given system using formal logics. The main advantage of this environment model is to assist in the construction, clarification, and validation of the given system requirements.

We have modelled an environment model for the heart for investigating the expected behaviours of a cardiac pacemaker. Given that a cardiac pacemaker interacts with the heart exactly at this level (i.e., electrical impulses), this model is a very promising "environmental model" to be used in parallel with a pacemaker model to form a closed-loop system. This model therefore has an immediate use in "the grand challenges in formal methods" where an industrial pacemaker specification has been elected as a benchmark. The closed-loop modelling of the heart and cardiac pacemaker involves formalizing and reasoning about pacemaker behaviour under normal and abnormal heart conditions.

A set of requirements is given in the closed-loop model for modelling general and patient specific conditions. Based on these requirements, we have presented an interactive and physiologically relevant closed-loop model for verifying basic and complex operations of a cardiac pacemaker. The main benefits of this work are as follows:

- To support verification and validation activities;
- Identifying gaps or inconsistencies in system requirements;
- Strengthen the given system requirements;
- To support "What-if" analysis during formal reasoning;
- Closed-loop modelling in early stage of system development;
- Analysing requirements, reducing cost, and virtualization;
- Increase confidence level and decrease the failure risks;
- To satisfy V&V requirements for domain specific certification standards;
- Automatic identification of new emergent behaviours;
- Validation of the system assumptions.

The main objectives of our work are to promote the use of such kind of closed-loop modelling approach to bridge a gap between software engineers and stakeholders to build a quality system, and to discover all ambiguous information from the requirements. Moreover, this approach helps to verify the correctness of behaviour of a system according to stakeholder requirements. There are scientific and legal applications as well, where the formal model based closed-loop modelling approach has certain scenarios to glean more information or better understandings of a system and assist to improve the final given system.

Acknowledgement. This work was supported by grant ANR-13-INSE-0001 (The IMPEX Project http://impex.loria.fr) from the Agence Nationale de la Recherche (ANR).

References

1. Bubenko Jr., J.A.: Challenges in requirements engineering. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, pp. 160–162, March 1995
2. McDermid, J.: Software Engineer's Reference Book. CRC Press Inc, Boca Raton (1991)
3. Lutz, R.: Analyzing software requirements errors in safety-critical, embedded systems. In: Proceedings of IEEE International Symposium on Requirements Engineering, pp. 126–133, January 1993
4. Davis, A.M.: Operational prototyping: a new development approach. IEEE Softw. **9**, 70–78 (1992)
5. Goguen, J., Meseguer, J.: Rapid prototyping: in the obj executable specification language. SIGSOFT Softw. Eng. Notes **7**, 75–84 (1982)
6. Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, RE 1995, pp. 140–147. IEEE Computer Society, Washington, DC (1995)

7. Noda, N., Kishi, T.: Aspect-oriented modeling for embedded software design. In: 14th Asia-Pacific Software Engineering Conference, APSEC 2007, pp. 342–349, December 2007
8. Karsai, G., Neema, S., Sharp, D.: Model-driven architecture for embedded software: a synopsis and an example. Sci. Comput. Program. **73**(1), 26–38 (2008)
9. Choi, K., Jung, S., Kim, H., hwan Bae, D.: Uml-based modeling and simulation method for mission-critical real-time embedded. In: System Development, IASTED Conference on Software Engineering 2006, pp. 160–165. Mittal, Zeigler and De la Cruz. (2006)
10. Kreiner, C., Steger, C., Weiss, R.: Improvement of control software for automatic logistic systems using executable environment models. In: Proceedings of 24th Euromicro Conference, vol. 2, pp. 919–923, August 1998
11. Auguston, M., Michael, J.B., Shing, M.T.: Environment behavior models for automation of testing and assessment of system safety. Inf. Softw. Technol. **48**(10), 971–980 (2006). Advances in Model-based Testing
12. Heisel, M., Hatebur, D., Seifert, T.S.D.: Testing against requirements using uml environment models. In: Fachgruppentreffen Requirements Engineering und Test, Analyse and Verifikation, pp. 28–31 (2008)
13. du Bousquet, L., Ouabdesselam, F., Richier, J.L., Zuanon, N.: Lutess: a specification-driven testing environment for synchronous software. In: Proceedings of the 21st International Conference on Software Engineering, ICSE 1999, pp. 267–276. ACM, New York (1999)
14. IEEE Standard: IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610. 12-1990, pp. 1–84, December 1990
15. Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide, 1st edn. Wiley, New York (1997)
16. Méry, D., Singh, N.K.: Formalization of heart models based on the conduction of electrical impulses and cellular automata. In: Liu, Z., Wassyng, A. (eds.) FHIES 2011. LNCS, vol. 7151, pp. 140–159. Springer, Heidelberg (2012)
17. Singh, N.K.: Using Event-B for Critical Device Software Systems. Springer, London (2013)
18. Méry, D., Singh, N.K.: Closed-loop modeling of cardiac pacemaker and heart. In: Weber, J., Perseil, I. (eds.) FHIES 2012. LNCS, vol. 7789, pp. 151–166. Springer, Heidelberg (2013)