Deliverable 4
Formal system engineering

28 juillet 2009 Version 1
February 1, 2010à 2:32 P.M.Version 2

Projet RIMEL

ANR-06-SETI-015

Equipe MOSEL, LORIA, Université Henri Poincaré Nancy 1
ClearSy
LABRI, Université de Bordeaux & CNRS
http://rimel.loria.fr
Années 2007-2008-2009

Avertissement
The following report has been written by Nazim Benaïssa, Thierry Lecomte, Dominique Méry, Joris Rehm and Neeraj Singh. Each chapter mentions his authors.

Contents

1 Présentation générale **7**
1.1 Introduction 8
1.2 Patrons de développement prouvé pour les systèmes 8
1.2.1 Raffinement automatique avec BART 8
1.2.2 Modélisation d'un système 9
1.2.3 Systèmes intégrant la sécurité 9
1.3 Conclusion 9

2 BART in action **11**
2.1 Introduction to Bart 12
2.1.1 Refined Elements 12
2.1.2 Refinement Rules 13
2.2 Refinement Rules Database 14
2.2.1 Variables refinement 14
2.2.2 Substitutions refinement 15
2.2.3 Database organisation 16
2.3 Application to Wayside Control Unit 17
2.3.1 System Description 18
2.3.2 Functional Specification 19
2.3.3 Added Refinement Rules 21

3 Modèle du temps et patrons **27**
3.1 Définition et usage des patrons 28
3.2 Exemple récurrent 30
3.3 Encodage des fonctions totales par des variables 30
3.4 Patron d'agenda absolu 31
3.5 Patron d'agenda relatif 34
3.6 Exemple 36
3.7 Patron de chronomètres 37
3.7.1 Modèle du patron 37
3.7.2 Insertion du patron raffiné 38
3.7.3 Représentation des contraintes temporelles 39
3.7.4 Représentation des propriétés temporelles 40
3.8 Conclusion 40

4 Applying patterns for modelling pacemaker-like systems **43**
4.1 Overview of Pacemaker System and Environment 44
4.1.1 The Heart Environment 44
4.1.2 The Pacemaker 44
4.2 Event-B Patterns 45
4.2.1 Action-Reaction Pattern 46
4.2.2 Time-Based Pattern 46
4.3 Overview of Pacemaker System Modelling 47
4.4 Abstract model of Pacemaker 49

4.4.1 Abstraction of AOO and VOO modes . . . 49
4.4.2 Abstraction of AAI and VVI modes . . . 51
4.4.3 Abstraction of AAT and VVT modes . . . 52
4.5 First refinement . . . 53
4.6 Second refinement:Threshold . . . 53
4.7 Third refinement:Hysteresis . . . 55
4.8 Fourth refinement:Rate Modulation . . . 57
4.9 Model Validation using ProB . . . 59
4.10 Conclusion . . . 59

5 Formal Development of Two-Electrode Cardiac Pacing System 61
5.1 Introduction . . . 63
5.2 Basic Overview of Pacemaker system . . . 64
5.2.1 The Heart System . . . 64
5.2.2 The Pacemaker system . . . 65
5.3 The modelling framework . . . 66
5.3.1 Modelling actions over states . . . 66
5.3.2 Model refinement . . . 67
5.3.3 Guidelines for EVENT B Modelling . . . 69
5.4 Formal Development . . . 70
5.4.1 The Context and Initial Model . . . 70
5.4.2 First refinement:Threshold . . . 94
5.4.3 Second refinement:Rate Modulation . . . 105
5.5 Model Validation and Analysis . . . 106
5.6 Conclusion and Future Works . . . 107

6 Adhoc systems 109
6.1 Introduction . . . 110
6.2 Overview of the modelling protocols . . . 111
6.3 Abstract model of basic communication protocol . . . 113
6.3.1 First Refinement . . . 116
6.3.2 Second Refinement . . . 118
6.3.3 Third Refinement : Route Discovery Protocol . . . 121
6.3.4 Fourth Refinement : Route Discovery Protocol . . . 122
6.3.5 Fifth Refinement : Route Discovery Protocol . . . 124
6.4 Conclusion . . . 126

7 SmartCards 127
7.1 Analysing cryptographic protocols . . . 128
7.2 Pattern for Modelling the Protocols . . . 128
7.2.1 The Pattern Structure . . . 130
7.2.2 An Example of the Pattern Application . . . 132
7.3 Event-B Models of the Mechanisms . . . 133
7.3.1 Abstract Model . . . 135
7.3.2 First Refinement . . . 137
7.3.3 Second refinement: attacker's knowledge . . . 137
7.4 Conclusion . . . 139

8 Self-healing systems 141

List of Figures

1.1 Bart en action 8

2.1 Bart schema 12
2.2 Bart refinement process order 12
2.3 Bart testing rule process 13
2.4 A Railroad Line of 5 Blocks 18
2.5 Different types of detectors related to a block 18
2.6 Functional analysis 19
2.7 Files generated by Bart 23
2.8 Status of the project 24

6.1 Ad-hoc Networks 113
6.2 Ad-hoc Networks of First refinement 116
6.3 Ad-hoc Networks of Fifth refinement 119

7.1 Hierarchy of authentication and key establishment goals 129
7.2 Hierarchy of authentication and key establishment goals 129
7.3 Abstract model 135

Chapter 1

Présentation générale

Sommaire

1.1 Introduction **8**

1.2 Patrons de développement prouvé pour les systèmes **8**

1.2.1 Raffinement automatique avec BART 8

1.2.2 Modélisation d'un système 9

1.2.3 Systèmes intégrant la sécurité 9

1.3 Conclusion **9**

1.1 Introduction

Le développement incrémental dirigé par la preuve de systèmes informatiques vise à mettre en œuvre la démarche de correction par construction (correct-by-construction)[81] dans le cadre de systèmes qui sont à logiciels prépondérants. Cette construction repose sur l'écriture de modèles événementiels dans le lanage de modélisation EVENT B et la progression de la construction est fondée sur le raffinement qui permet de préciser de plus en plus ce qui sera un modèle final. cette démarche progressive est validée par la relation de raffinement mais aussi par la preuve mathématique de conditions de vérification permettant de garantir que le modèle concret satisfait les mêmes propriétés que le modèle abstrait. La démarche générale consiste donc à expliciter un modèle abstrait qui répond bien au problème posé et que le concepteur va progressivement modifier pour le rendre de plus en plus proche de la solution finale visée. Cette solution finale peut être un algorithme séquentiel, un algorithme réparti ou un système à logiciel prépondérant. Une des idées développées dans le cadre de ce projet est celle de patron de conception prouvé et cette idée se fonde sur des observations faites au cours des études de cas réalisées pour construire des modèles de systèmes. Jean-Raymond Abrial[59] a souligné l'intérêt d'une telle démarche capable de capitaliser des preuves parfois difficiles mais qui sont réutilisables dans d'autres dévelopements. Une telle approche avait été suivie dans le cadre du raffinement automatique avec la première version du système BART par notre partenaire industriel et la société SIEMENS pour le développement d'un système de coàntrôle pour un métro à Paris. Dans un cadre industriel, l'importance est mise sur la diminution des coût tout en préservant la qualité du produit développé. Nous avons déjà proposé un certain nombre de développements dans le cadre des livrables préccédents [18, 19, 20]. Pour ce livrable, il nous est paru important de mettre en évidence les patrons que l'on pouvait réutiliser effectivement et aussi d'étendre notre champ d'activité aux systèmes à logiciel prépondérant. La notion de patron est assez répandue mais ne recouvre pas toujours les mêmes idées et le langage de modélisation EVENT B n'échappe pas à ce constat. Nous avons considéré plusieurs domaines d'applications et essayer de donner des définitions claires de ce qu'est un patron fondé sur la preuve.

1.2 Patrons de développement prouvé pour les systèmes

1.2.1 Raffinement automatique avec BART

Les études menées dans les tâches ont conduit à des réalisations concrètes avec BART [23] qui est une redéfinition de cet outil dans le cadre de l'AtelierB 4.0. Cette version de l'AtelierB est différente de la distribution RODIN mais permet de développer aussi des modèles EVENT B .

L'environnement BART propose d'engendrer des modèles B automatiquement en appliquant des règles de raffinement. Les modèles engendrés sont vérifiés par l'outil. L'approche suivie par BART est décrite dans le chapitre 2 et a été utilisée dans des développements industriels. BART est un outil qui permet de raffinement automatqieument des composants B et ce processus est fondé sur des règles de raffinement proposés par le développeur. Un langage de règles spécifiques a été défini et BART r(affine au moins une machine. La figure 1.1 décrit le processus opéré. Dans le cas de BART, l'objectif estd e fournir des machines d'implantation qui peuvent ensuite conduire à du code C/C++/ADA etc

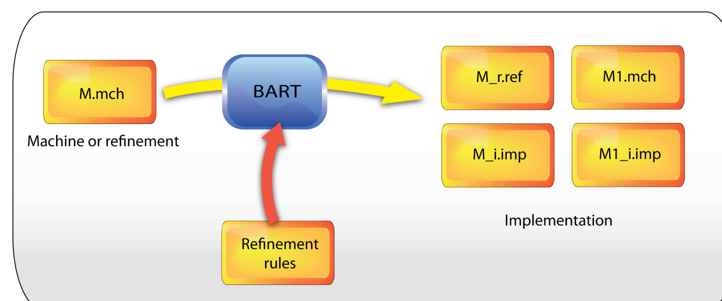


Figure 1.1: Bart en action

1.2.2 Modélisation d'un système

Deux patrons ont été introduits indépendamment et portent sur la modélsation des systèmes réactifs. Il s'agit d'une part du patron action/réaction de Jean-Raymond Abrial[?] introduit dans le cadre du développement d'une presse et d'autre part du patron permettant d'ajouter du temps au modèle dû à J. Rehm présenté dans le chapitre 3. Un patron dans ce contexte est un guide méthodologique fondé sur des éléments de modélisation à prendre en compte comme le temps ou l'action/réaction. Il s'agit d'un élément important du système à modéliser et le patron apporte un guide et une aide sur ce qu'il convient d'ajouter au modèle courant. Ces deux techniques ont été développées séparément mais ont été utilisées dans une modélisation d'un pacemaker 4. Le système est analysé suivant ses modes de fonctionnement et les deux patrons sont combinés. Le travail de modélisation a été facilité par l'utilisation de ces patrons de conception prouvés et cette utilisation a permis de monter que ces patrons étaient réutilisables dans des études de cas différentes des études ayant conduit à leur énoncé.

1.2.3 Systèmes intégrant la sécurité

Les travaux sur les algorithmes cryptologiques ont permis de mettre en évide,ce un patron de développement prouvé permettant d'intégrer des élements dans les modèles afin de valider telle ou telle propriété propre aux algorithmes cryptologiques. Le chapitre 7 apporte une description assez complète de cette technique.

1.3 Conclusion

Ce livrable fait le point sur ce que nous avons appelé le développement de systèmes par la méthode incrémentale validée par la preuve. Les systèmes considérés dans ce livrable sont le plus souvent liés à des problèmes algorithmiques de la répartition complexes comme les systèmes intégrant la cryptologie. Le pacemaker est un autre type de système qui a permis d'illustrer les patrons mis en évidence par J.-R. Abrial et J. Rehm. Le cas de l'algorithme de Dijkstra est différent puisqu'il s'agit en fait de l'autostabilisation qui est une propriété très difficile à démontrer. Ce point reste à approfondir car il y a des éléments méthodologiques à produire à partir de l'exemple développé dont nous avons donné uniquement les modèles Event B. Concernant le lien avec les outils, nous avons en chantier des applications mettant en œuvre les patrons. Enfin, un patron permet de partager à d'autres l'expérience que nous avons gagné lors de nos études de cas; le groupe a longuement échangé sur cette notion et la perçoit comme une aide. La définition formelle d'un patron reste à développer mais l'idée générale est sans doute plus précise. On notera que cette définition est liée à des études de cas et que comme pour les objets, il faut développer pour bin comprendre ce concept. Le groupe poursuit donc vers une définition plus formelle. Les mois à venir seront employés pour publier ces résultats.

Chapter 2

BART in action

Ce chapitre a été rédigé par Thierry Lecomte.

Sommaire

2.1 Introduction to Bart 12
2.1.1 Refined Elements 12
2.1.2 Refinement Rules 13
2.2 Refinement Rules Database 14
2.2.1 Variables refinement 14
2.2.2 Substitutions refinement 15
2.2.3 Database organisation 16
2.3 Application to Wayside Control Unit 17
2.3.1 System Description 18
2.3.2 Functional Specification 19
2.3.3 Added Refinement Rules 21

2.1 Introduction to Bart

BART is a tool that can be used to automatically refine B components. This process is rule based so that the user can drive refinement. Its own rule language has been defined in this purpose. It has been designed to be a stand-alone tool, but it may be launched from Atelier B user interface. As an input, Bart must be given at least the machine or refinement to refine. There must be exactly one component to refine.

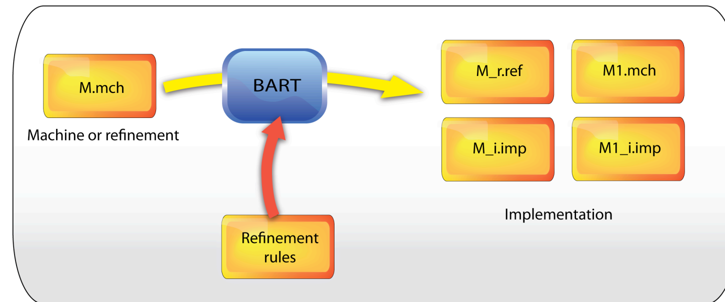


Figure 2.1: Bart schema

Automatic refinement is a rule based refinement process for B components (abstractions or refinements). The tool is given a component, and it searches, for each element to refine, some rules that specify how it must be treated. These rules allow to implement design patterns for B models. Bart is specialized for B software models but is likely to be extended to support event B models.

2.1.1 Refined Elements

First elements processed by the Bart tool are the abstract variables of the component to refine (content of the $ABSTRACT_VARIABLES$ clause).

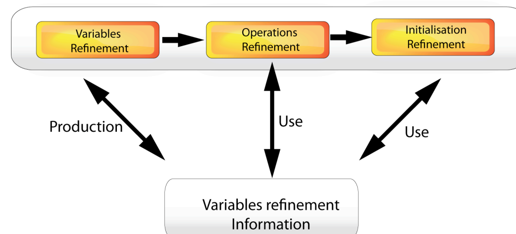


Figure 2.2: Bart refinement process order

The tool must produce, for each one of them, one or more abstract or concrete variables that implement it.
Bart processes operations of given component in order to refine them. It must produce, for each operation, a substitution body concrete enough to be put in the component implementation. Refined operations are considered for the whole component abstraction. It means that Bart refines the most concrete version of each operation.
Bart also refines content of initialisation clause of given component. Typically, it produces a concrete result by specifying initialisation substitutions for concrete variables refining content of $ABSTRACT_VARIABLES$ clause.
Abstract variables are refined first, as other parts of the process need its output to find suitable rules for operations and initialisation. It is necessary at these steps to know how variables have been refined. This variable information is stored as predicates in Bart hypothesis stack.
As it will be described later, refinement process uses rules to determine how each element is refined. A same rule can apply for several elements, so it must be general. In this purpose, the rule language uses jokers, so that rules can contain variable parts.

2.1.2 Refinement Rules

Bart uses rules for refining variables, operations and substitutions. These rules belong to different types:

- variables rules, or
- substitution rules, which can be used for both operations and initialisation.

Rules of same type are gathered in theories (ako packages). Theories are associated to predicates/expressions families, as referred in the tactics theory. Rules usually contain a pattern, and may contain a constraint. These two elements are used to know if a rule can be applied to refine a certain element. Rules also contain clauses that express the refinement result.
Rules may have constraints, expressed in their $WHEN$ clause. A constraint is a predicate, which may contain jokers. It may be a complex predicate, built with "&" and "OR" operators. Bart contains a stack of hypothesis , which is built from the machine to refine and its environment. A constraint is successfully checked if its elementary elements (element not containing "&" or "OR") can be pattern-matched with a predicate of the stack so that the complex constraint is true.

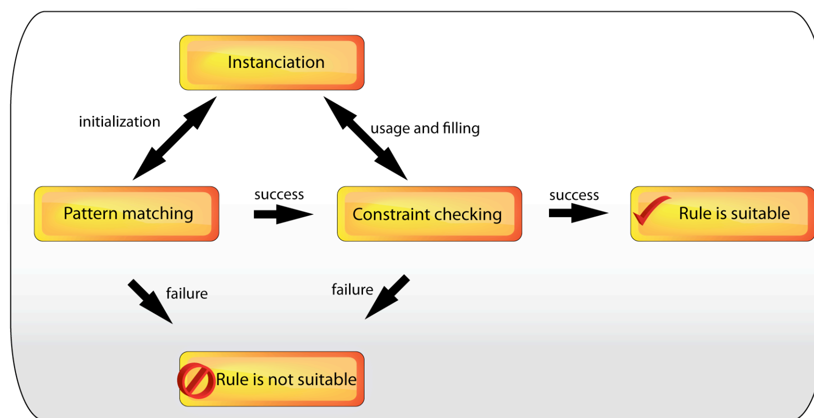


Figure 2.3: Bart testing rule process

According to operators, Bart uses backtracking to try every combination of instantiation that should be a success. If several instantiations can make the constraint be successfully checked, Bart uses one of them. In this case, it is better to write a more detailed constraint to have only one result. If there are several results, Bart could choose one which is not the one the user had planned. Usually, when checking a constraint, some jokers have already been instantiated.
Guards are special predicates which may be present in rule constraint clauses. They allow checking some properties on elements to refine and their environment.
There are two kinds of guards:

- some are simply present in the predicate stack. They are added at the environment loading. For instance $ABCON$ (abstract constant) and $ABVAR$ (abstract variable) belong to this kind of guards.
- the other kind is calculated guards. For these ones, during constraint checking, Bart doesn't try to match them) with the stack, but directly calculates if the guard is true or false. These kinds of guards may also have side effects. For example bnum (numeric test) or bident (identifier test) are calculating guards.

Guards are simply put in the constraint as regular predicates.
This process is used for variables, operations and initialisation refinement, although it is simpler for variables. Every rule contains a pattern. First Bart tries to match it with the element to refine. If it succeeds, it tries, if a rule has a constraint clause, to match the predicate against hypothesis. When checking the constraint, some jokers have already been instantiated by pattern matching. If the constraint checking is a success or the rule had no constraint, then it will be used to refine current element.

Variable process is simpler as variable rules have simple pattern, which is a single joker. Variable rule patterns are only matched in order to instantiate the joker representing currently refined abstract variable. This joker is reused in WHEN or result clauses.

2.2 Refinement Rules Database

Rule files are files containing theories, each theory containing one or several rules used to refine given component. Rule file extension is usually .rmf. A rule file can contain variable, operation, structure and initialisation theories. It can also contain utility theories such as tactic, user pass, or definition of predicates synonyms.
The rule file syntax must also respect certain constraints:

- User pass can be present at most once
- Tactic can be present at most once
- Predicate theory can be present at most once

Order between theories has no syntactical impact, expect for predicates theory: it must be defined before its elements are used in the rule. Order between theories has an impact on the rule research, as the standard process (no user pass or tactic) reads theories from bottom to top. User pass and tactic can be defined anywhere in the file, even before theories they refer to have been defined.
The Bart tool comes with a set of predefined rule base, contained in the file PatchRaffiner.rmf present in Bart distribution. It provides rules that permit to refine most of the classical B substitutions.
The classical automatic refinement scheme is the following: most elements of given component can be refined using the rule base. If an element can not be refined with it, or needs a more specific treatment, user should write suitable rules in rmf files that will be provided after the rule base on command line, or in the component associated rule file.

2.2.1 Variables refinement

Variable rule research is different from rule research for operations and initialisation. Instead of processing each variable and finding a suitable rule for it, it processes each rule of considered theories (all variable theories or a subset if tactic or user pass is used) and checks if it can be used to refine some variables. This is necessary because a single variable rule can be used to refine several variables. Once a rule has been selected for one (or several) variable, resulting refinement variables can be calculated from its clauses.
The principle of rule research is the following:

- At the beginning the tool considers the set of abstract variables to refine
- It processes every theory that could be used (according to tactic, user pass or neither) from bottom to top. For each theory:
  - The tool processes all rules of theory from bottom to top. For each variable rule:
    - Bart determines which variables can be refined by current rule
    - Refined variables are removed from the set of remaining variables

This process stops when there are not variables to refine anymore, or when all variable rules to consider have been treated. Variable refinement is successful if all variables have been associated with a rule. It is a failure if all rules have been treated and some variables could not be refined.
For a certain rule, Bart determines which variables it can refine as follow:

- The tool tries every combination of values to instantiate joker list of $VARIABLES$ clause. For each instantiation:
  - Bart checks constraint expressed in WHEN clause against hypothesis stack, with jokers of $VARIABLES$ clause instantiated

* If $WHEN$ constraint could be checked, variables used to instantiate $VARIABLES$ clause can be refined by this rule
* Variable refined by the rule are removed from set of remaining variables, to be sure they won't be used in following tried instantiation

If current rule has several jokers in $VARIABLES$ clause, there are more combinations to try than for simple variable rules.

2.2.2 Substitutions refinement

Substitution refinement gathers operation, initialisation and structural rules. Operation and structure rules are identical. Initialisation rules are simpler versions of operation rules. Substitution refinement is more complex than variable refinement, as it can be recursive, i.e. result of refinement for a given substitution may have to be refined too. Furthermore, for a given substitution, refinement may need several sub-processes (use of $SUB_REFINEMENT$ clause or default refinement behaviours for parallel or semicolon). So refinement sub-branches are created and the underlying structure that can be used to represent substitution refinement is in fact a tree.
The substitution rule research process is simpler than for variables.

- For a substitution to refine, Bart processes each rule file as long as he could not find a rule.
- For each rule file it processes operation theories to consider (all theories, or a subset if tactic or user pass is used) from bottom to top.
- For each theory it processes operation rules from bottom to top
- For each rule, Bart checks if it can be used to refine currently treated substitution.

If each rule file was processed by Bart and no rule could be found for a certain substitution, an operation refinement may occur. Each operation rule has a pattern ($REFINES$ clause) and may have a constraint ($WHEN$ clause).
The substitution refinement process depends, for given rule and substitution, on the presence and content of $SUB_REFINEMENT$, $IMPLEMENTATION$ and $REFINEMENT$ clauses.
$SUB_REFINEMENT$ clause contains a comma-separated list of sub-elements. Each sub-element left part is a substitution that may contain jokers. These jokers must all have been instantiated by pattern matching and constraint checking. Right part of the sub-element must be a single and still uninstantiated joker. This clause is used to refine the given substitution and store the result in given joker. This is done before calculation of the rule substitution result, so the sub-refinement can be used to express the result.
$IMPLEMENTATION$ clause expresses the result of current rule. It contains a substitution which may contain jokers. All these jokers must have been instantiated during pattern matching, constraint checking or sub-refinement processing. $IMPLEMENTATION$ clause may also contain concrete operation refinement variable declaration. Using $IMPLEMENTATION$ clause means that given result is the final result of current branch and does not need to be refined again.
$REFINEMENT$ clause expresses the result of current rule. It contains a substitution which may contain jokers. All these jokers must have been instantiated during pattern matching, constraint checking or sub-refinement processing. $REFINEMENT$ clause may also contain abstract or concrete operation refinement variable declaration. Using $REFINEMENT$ clause means that given result is not the final result of current branch. The result of rule must be refined. $IMPLEMENTATION$ and $REFINEMENT$ clauses can not be both used in a same rule. When a rule has been selected (and eventual sub-refinements have been processed), the rule result is calculated by instantiating jokers of its result clause.
A rule can contain both $SUB_REFINEMENT$ and $REFINEMENT$ clauses. In this case, each subrefinement is calculated and stored in its joker. Then content of $REFINEMENT$ clause is instantiated and refined. For a substitution to refine, if no rule could be found, Bart will check if it can be refined using a predefined behaviour. For some kinds of substitutions, Bart may know how to refine them if no rule is present. Predefined behaviour can be the end of current branch (skip substitution refinement) or a simple node of refinement tree. In this case, Bart may create one ($BEGIN$ substitution refinement) or several (semicolon refinement) subnodes in refinement tree for current substitution. For each new subnode created by predefined refinement behaviour, the recursive refinement process is restarted as a rule or predefined behaviour will be searched for each one.

2.2.3 Database organisation

The database that is coming along with Bart is a sample database, composed of 228 rules, requiring to be adapted and extended to particular needs. It is provided mainly to show how such a set of rules can be constituted, as it would be quite difficult for a newcomer to develop such a database from scratch. In short, do not expect to fully automatically refine any B model with it, without any modification/addition.
This database is composed of 28 theories, decomposed as follows:

- 1 theory for predicates: this theory contains 1 definition that is used by most rules. The rule:

  ```
  B0(@a) <=> (B0EXPR(@a) or SCALAR(@a))
  ```

  defines that a parameter `@a` is $B0$-compliant if it either a `B0` expression or a `SCALAR`.

- 2 theories for variables: these theories contain 4 rules for refining standard variables and producing iterators. For example, the rule:

  ```
  RULE scalar
  VARIABLE
      @a
  TYPE
      SCALAR(@a)
  WHEN
      (PR(@a : BOOL) &
          match(@b, BOOL)) or
      (PR(@a : INTEGER) &
          match(@b, INT)) or
      ((SET(@b) or ENUM(@b))&
          PR(@a : @b))
  IMPORT_TYPE
      @a : @b
  CONCRETE_VARIABLES
      @a
  INVARIANT
      @a : @b
  END
  ```

  allows to refine an abstract variable (of type `BOOL`, `INTEGER` or `ENUM`) into a concrete one of the same type and same name. The tag `SCALAR(@a)` is added to the predicate stack.

- 2 theories for initialisations: these theories contain 9 rules for refining standard variables and iterators initialisation. For example, the rule:

  ```
  RULE scalar_ini2
  REFINES
  @a :: @b
  WHEN
  SCALAR(@a) &
  PR(0 : @b)
  IMPLEMENTATION
  @a := 0
  END
  ```

  allows to implement the non-deterministic substitution `@a :: @b` as `@a := 0` if 0 belongs to `@b`.

- 20 theories for operations: these theories contain 153 rules. For example, the rule:

```
RULE assign_a_b_6
REFINES
    @a := @b
WHEN
    B0(@a) &
    DECL_OPERATION(@c <-- @d | BEGIN @c := @b END)
IMPLEMENTATION
    @a <-- @d
END
```

implements the substitution `@a := @b` (where `@a` is B0-compliant), by calling the operation `@d` declared as:

```
@c <-- @d =
    BEGIN
        @c := @b
    END
```

in a $SEEN$ component.

- 2 theories for structures: these theories contain 25 rules. For example, the rule:

```
RULE any_1
REFINES
    ANY @a WHERE
        @b
    THEN
        @c
    END
REFINEMENT
    @a :( @b );
    @c
END
```

  refines an `ANY` substitution into the sequence of two substitutions:

  - not deterministic choice for `@a`
  - execution of the substitution `@c`

- 1 theory for tactics: this theory is the main switch of the database, as it defines the mapping between predicates and theories (what theories have to be applied on a particular predicate). For example, the rule:

```
becomes_member_a_b => (@a :: @b)
```

  associates the theory $becomes_member_a_b$ to the predicates matching $@a :: @b$. This theory contains 1 rule for variable refinement, 1 rule for initialisation refinement and 34 rules for operation refinement.

2.3 Application to Wayside Control Unit

In order to assess the tool, we have decided to apply Bart on the development of a part of the Wayside Control Unit of the Roissy Airport Val Shuttle. This kind of system is safety critical (Safety Integrity Level 3). The related software was previously developped by ClearSy, by using Siemens Transportation Systems own automatic refiner and would provide a good reference for assessing the tool.
In the following sections, the overall system is presented, as well as the functional specification of the software being developped. Then the process of extending the refinement rules database is exposed, as well as the resulting generated B models.

2.3.1 System Description

A railroad line is supposed to be divided into fixed blocks. The line has two directions 'up' and 'down'. Each block may only be connected to one upward block and to one downward block. So the line is quite simple, since it has no switch. Figure 2.4 gives an example of such a line with 5 blocks.

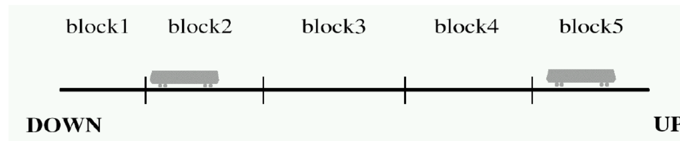


Figure 2.4: A Railroad Line of 5 Blocks

Actually, this system is the simplification of a more realistic one handling switches.
Trains may drive up-bound or down-bound on the railroad line and they may change direction at any time. The purpose of the functionality developed in this exercise is to establish safely, from the software point of view, which blocks are occupied by a train and which are free. Here are the basic principles given by the system analyses. For each block a detector located along the track called Trackside Detector (TD) is used to detect trains. A train is equipped with an antenna located below the coach. When the antenna is above a trackside detector, a signal inside the detector is produced, so the train presence may be detected. Now building the software appears to be easy, since we just need to read for each block the state of its trackside detector. However, this solution raises two issues, both related with safety:

- The information given by a trackside detector is not accurate enough on the border of detectors.
- Trackside detectors (or antennas) may be faulty.

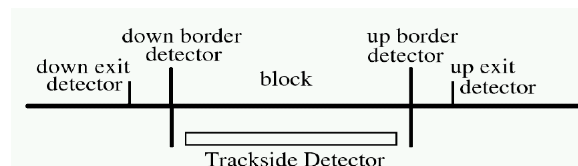


Figure 2.5: Different types of detectors related to a block

To overcome those issues, the following elements are added to the system specification:

- A Border Detector (BD) is used at each borderline between two blocks to achieve accurate block occupancy detection. When a block trackside detector or border detector is occupied, then the block is considered to be occupied.
- Exit Detectors (ED) located after a block border (in the upward block or in the downward block) are used to detect trains leaving the block. A block is considered to be released on the falling edge of one of its exit detector.
- The Trackside Detector Loss (TDL) alarm is set for a block when a trackside detector inconsistency happens. Such an inconsistency happens when a block trackside detector is free although it should be occupied. When a TDL alarm is set, the procedure to release it requires that an operator at the command center to send back an alarm acknowledgement.
- To avoid unjustified TDL alarm due to the lack of accuracy of trackside detectors, blocks may be masked for TDL alarm when trains are located near a block border.

2.3.2 Functional Specification

The original software, developped years ago, is quite large, representing around 180 000 lines of code [1]. Redeveloping such a software is out of the scope of this project, so it was decided to re-engineer only one function: the block function, in charge of determining which blocks are empty and which blocks are occupied. The position of this function within the software is presented on Figure 2.6:

- the input function provides track information to the block function
- the route logic allocates blocks to a train in order to create a route
- the mode logic function determines which mode the train is running (nominal, degraded, faulty)
- the output function provides authorization information

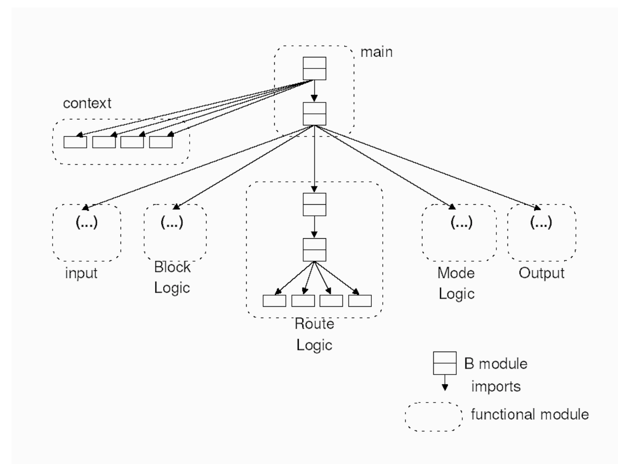


Figure 2.6: Functional analysis

This information is critical, as authorizing a train to enter an occupied block may likely lead to a collision between the two trains. The specification of the block function provided here is based on a real example, although it has been simplified.
This software controls a railroad line, divided into fixed blocks. The purpose of the functionality developed here is to manage safely block occupancy by trains. This functionality is composed of 6 services that are described below and which are given a formal specification in B.

- release_tdl_alarm: This function releases Trackside Detector Loss (TDL) alarm. When a TDL alarm acknowledgment is received from the Control Center, then TDL alarm is released for all blocks. If no TDL alarm acknowledgment is received then TDL alarm remains unchanged.

```
IF  cc_tdl_ack = TRUE THEN
tdla := {}
END
```

- set_tdl_alarm: This function sets Trackside Detector Loss (TDL) alarm. When a block does not become in TDL alarm, then the alarm remains unchanged. A block becomes in TDL alarm, when the following conditions are true:
    1. The block is occupied.
    2. The block is not masked.
    3. The block trackside detector is free.

[1] see http://www.clearsy.com/pdf/val-roissy.pdf

```
tdla := tdla \/ (ob - mb - otd)
```

- occupy_blocks: This function manages occupied blocks. Blocks that do not become occupied remain unchanged. A block is consider to be occupied when one of its border detector is occupied or when its trackside detector is occupied.

- unmask_blocks: This function unmasks some blocks (for TDL alarm). Blocks which do not become unmasked remain unchanged. A block is unmasked when the block is free or when all of the following conditions are true:

  1. The upward block has a free trackside detector or the upward block is free.
  2. The downward block has a free trackside detector or the downward block is free.

```
mb := mb -(d_free_b \/ (cfg_b2b_up~  [d_free_td \/ d_free_b]
                             /\ cfg_b2b_down~[d_free_td \/ d_free_b]))
```

- mask_blocks: This function masks some blocks (for TDL alarm). A block is masked when the following conditions are true:

  1. The block is not in TDL alarm.
  2. One of the block borders is occupied.

  Blocks that do not become unmasked remain unchanged.

```
mb := mb \/ (d_bd2b[obd] - tdla)
```

- release_blocks: This function manages released blocks. Blocks that are not released remain unchanged. A block is considered to be released when the following conditions are true:

  1. The block is not in TDL alarm or the block is being initialized by the Control Center.
  2. A block exit detector, which was occupied during the previous cycle, is now released.

```
PRE
   p_block : t_block
THEN
   SELECT
      p_block : cc_init &
      p_block : (cfg_b2ed_up \/ cfg_b2ed_down)~[oed_prev - oed]
   THEN
      ob := ob - {p_block}
   WHEN
      p_block /: tdla &
      p_block : (cfg_b2ed_up \/ cfg_b2ed_down)~[oed_prev - oed]
   THEN
      ob := ob - {p_block}
   ELSE
      skip
   END
END
```

2.3.3 Added Refinement Rules

To be able to refine automatically the specification of these 6 services, only 6 rules were added to the project. These new rules are not directly added to the patchrefiner, but inserted into a new file that would complement the existing patchrefiner. 5 of these rules are $OPERATIONS$ rules and 1 is a $VARIABLES$ rule.
These rules were designed during the interactive refinement process, where the user can determine why a variable or an operation is not completely refined/implemented. If you have a look at the above services specifications, you can see specific substitutions that would require specific refinement rules.
For example, the rule:

```
RULE R3
REFINES
    @a := @a - ({@b} /\ @c)
IMPLEMENTATION
    IMPORTED_OPERATION (
        out  => (#1),
        in   => (),
        pre  => (0 = 0),
        body => (#1 := bool (@b : @c))) ;
    IF #1 = TRUE THEN
        IMPORTED_OPERATION (
            out  => (),
            in   => (),
            pre  => (0 = 0),
            body => (@a := @a - {@b}))
    END
END
```

implements the substitution `@a := @a - ({@b} /\ @c)` with the sequential call of:

- an operation which returns a boolean indicating if `@b : @c` holds
- a test evaluating the value of the boolean computed at the previous step
- an operation substracting `{@b}` from `@a`

The lines

```
      IMPORTED_OPERATION (
           out  => (),
           in   => (),
           pre  => (0 = 0),
           body => (@a := @a - {@b}))
```

lead to the creation of an operation with no input and no output parameter, but executing the substitution `@a := @a - {@b}`.
The following rule

```
RULE R1
REFINES
    @a := bool(@g : (@b \/ @c)~[@d])
WHEN
    @b : @e +-> @f &
    @c : @e +-> @f &
    B0EXPR(@g) &
    DECL_OPERATION(@h, @i <-- @j(@k) |
        PRE
            @r
        THEN
            @h := bool(@k : dom(@b)) ||
```

```
                IF @k : dom(@b)
                THEN
                    @i := @b(@k)
                ELSE
                    @i :: @l
                END
            END) &
        DECL_OPERATION(@m, @n <-- @o(@p) |
            PRE
                @q
            THEN
                @m := bool(@p : dom(@c)) ||
                IF @p : dom(@c)
                THEN
                    @n := @c(@p)
                ELSE
                    @n :: @s
                END
            END)
    IMPLEMENTATION
        #1,#2 <-- @j(@g);
        IF #1 = TRUE
        THEN
            IMPORTED_OPERATION (
                out  => (@a),
                in   => (#2),
                pre  => (#2 : @f),
                body => (@a := bool(#2 : @d)))
        ELSE
            @a := FALSE
        END;
        #1,#2 <-- @o(@g);
        IF #1 = TRUE
        THEN
            IMPORTED_OPERATION (
                out  => (#1),
                in   => (#2),
                pre  => (#2 : @f),
                body => (#1 := bool(#2 : @d)))
        END;
        @a := bool(#1 = TRUE or @a = TRUE)
    END
```

is more complex than the previous one. The substitution being refined is `@a := bool(@g : (@b \/ @c)~[@d])`. If

- `@b` and `@c` are two partial functions
- `@g` is a B0-compliant expression
- `@j` and `@o` are two operations with one input parameter, two output parameters and complying with the specified body

then a double test is performed to check wether `@g : @b~[@d]` or `@g : @c~[@d]`.
Bart also provides some traces, indicating which rules have been applied. Variables refinement is quite straightforward:

```
Variables refinement.
```

```
  --- Refinement rules ---
    Variable: mb, Rule: standard.setArray
    Variable: ob, Rule: standard.setArray
    Variable: tdla, Rule: standard.setArray
```

For operations refinement, more messages are generated by the tool:

```
Operations refinement.
  -- Refinement of operation mask_blocks.
      assign_a_union_b_c.assign_a_union_b_c_1

  -- Refinement of operation read_ob.
      assign_a_bool_belongs_b_c.assign_a_bool_belongs_b_c_39

  -- Refinement of operation release_blocks.
      select.select_with_else_split_cond_when
      default.if_then_else_0
      default.default
      default.if_then_else_1
      select.select_with_else_test_cond_when
      default.if_then_else_0
      default.default
      default.if_then_else_1
      default.default
      select.select_with_else_simplify_cond
      select.select_with_else_test_cond
      default.if_then_else_0
      default.default
      default.if_then_else_1
      default.default
      select.select_with_else_test_cond
      default.if_then_else_0
      default.default
      default.if_then_else_1
      default.default
(...)
```

We can see at that occasion the rules applied on the refinement of the `ANY ... WHERE ... THEN ... END` substitution.

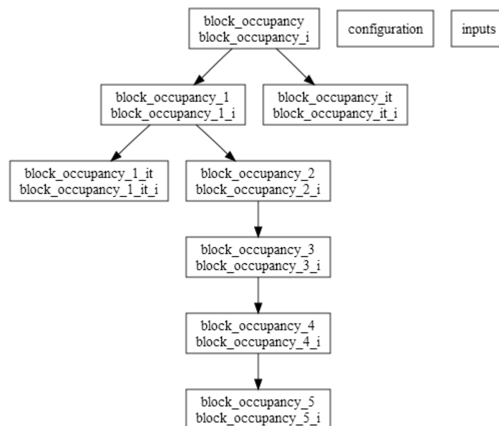


Figure 2.7: Files generated by Bart

Below is the contents of one the components generated by the tool:

```
MACHINE
  block_occupancy_1
SEES
  configuration ,
  inputs
ABSTRACT_VARIABLES
  mb , ob , tdla
INVARIANT
  mb <: t_block_i &
  ob <: t_block_i &
  tdla <: t_block_i
INITIALISATION
  mb := {} ||
  ob := t_block ||
  tdla := t_block
OPERATIONS
  mask_blocks_1(par_in_0_1) =
  PRE
    par_in_0_1 : t_block_i &
    par_in_0_1 : t_block
  THEN
    mb := mb \/ ({par_in_0_1} /\ ((cfg_b2bd_up \/ cfg_b2bd_down)~)[ obd] - tdla)
  END;

...

  unmask_blocks_1(par_in_0_1) =
  PRE
    par_in_0_1 : t_block_i &
    par_in_0_1 : t_block
  THEN
    mb := mb - ({par_in_0_1} /\ (t_block - ob \/ ((cfg_b2b_up~)[ t_block - otd \/ t_bloc
  END
END
```

Finally, all the components of the project are demonstrated by proof. This way, we are validating at the same time the tool and the refinement rules database.

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
block_occupancy	OK	OK	8	8	0
block_occupancy_1	OK	OK	7	7	0
block_occupancy_1_i	OK	OK	33	33	0
block_occupancy_1_it	OK	OK	6	6	0
block_occupancy_1_it_i	OK	OK	16	16	0
block_occupancy_2	OK	OK	5	5	0
block_occupancy_2_i	OK	OK	2	2	0
block_occupancy_3	OK	OK	5	5	0
block_occupancy_3_i	OK	OK	31	31	0
block_occupancy_4	OK	OK	4	4	0
block_occupancy_4_i	OK	OK	4	4	0
block_occupancy_5	OK	OK	4	4	0
block_occupancy_5_i	OK	OK	37	37	0
block_occupancy_i	OK	OK	40	40	0
block_occupancy_it	OK	OK	6	6	0
block_occupancy_it_i	OK	OK	16	16	0
configuration	OK	OK	0	0	0
inputs	OK	OK	4	4	0

Figure 2.8: Status of the project

Ce chapitre a été rédigé par Joris Rehm.

Chapter 3

Modèle du temps et patrons

Sommaire

3.1 Définition et usage des patrons **28**
3.2 Exemple récurrent **30**
3.3 Encodage des fonctions totales par des variables **30**
3.4 Patron d'agenda absolu **31**
3.5 Patron d'agenda relatif **34**
3.6 Exemple **36**
3.7 Patron de chronomètres **37**
3.7.1 Modèle du patron 37
3.7.2 Insertion du patron raffiné 38
3.7.3 Représentation des contraintes temporelles 39
3.7.4 Représentation des propriétés temporelles 40
3.8 Conclusion **40**

Nous allons introduire dans ce chapitre notre modélisation du temps, ces concepts nous permettront d'étudier les propriétés temps-réel des systèmes. Cette modèlisation du temps est décrite sous la forme de patrons qui permettent de partager à d'autres l'expérience que nous avons gagné lors de nos études de cas.
Notre motivation est d'incorporer un modèle du temps dancs les modèles B évènementiels, ce qui permet d'étudier des systèmes temporels en conservant le langage et les outils standard.
L'idée générale est de réaliser une modélisation explicite du temps : des variables sont ajoutées pour exprimer l'état temporel du système. La progression du temps est exprimée par un évènement (nommé tic), qui est le seul à faire varier les variables temporelles (et qui ne modifie pas les autres variables du système). Il est aussi possible de concevoir un modèle ou la progression du temps est effectuée en parallèle avec les transitions du système, mais ceci est plus délicat à exprimer et se révèle inadapté dans le cas d'un système distribué (puisque que l'on peut avoir plusieurs transitions d'état dans le même instant). Comme les autres évènements d'une machine temporisée ne doivent pas modifier les variables temporelles, il s'ensuit que ces évènements sont instantanés. Ceci est compatible avec le concept des évènements qui sont des actions atomiques et discrètes. Donc lorsque que l'on veut représenter une opération qui dure dans le temps, il faut alors introduire deux évènements, un pour le début et l'autre pour la fin de l'opération (et contraindre le temps qui s'écoule entre ces deux évènements).
Nous allons d'abord définir ce que nous entendons par patron. Puis nous présentons notre patron d'agenda absolu puis relatif et enfin notre patron de chronomètre. Ces trois patrons seront appliqué sur un exemple simple qui sera introduit après la définition des patrons.

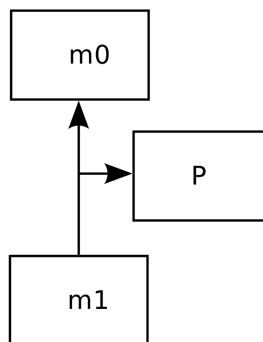
3.1 Définition et usage des patrons

Un patron est un modèle représentant une solution pré-étudiée et générique à une tâche de conception ou d'étude d'un phénomène.
Un patron devrait être créé lorsque l'on observe des comportements similaires ou des taches répétitives. Le patron doit alors proposer une solution générique que l'on peut réutiliser dans les différents cas du phénomène identifié. Cette solution est générique mais l'appliquer demande néanmoins un effort car les problèmes considérés font souvent preuve d'une grande variété de forme.
La motivation est de pouvoir faire face efficacement à des situations complexes. Il s'agit aussi de travailler de manière méthodique et de rendre le développement plus systématique. Les avantages attendus sont de travailler plus vite et d'obtenir un résultat de meilleur qualité (moins d'erreur, plus structuré, maintenable et analysable).
Étymologiquement un patron est un modèle sur lequel on fabrique des objets. Il peut s'agir par exemple d'une partie de vêtement mais aussi d'une forme à peindre ou d'une pièce de bois en lutherie.
Un usage notable des patrons à été fait en 1977 concernant l'architecture (des bâtiments). Le livre [**?**] propose un vaste catalogue de solutions, à la conception de bâtiments, décrites dans un vocabulaire codifié.
Et bien sûr en informatique, plus précisément dans le domaine de la conception objet, l'usage des *design pattern* (patron de conception) constitue un phénomène important depuis la parution de [**?**].
En allant plus loin dans la définition des patrons, on peut remarquer qu'ils proposent un niveau d'abstraction supplémentaire par rapport au langage usuel du domaine considéré. Par exemple, dans le monde de la conception objet, on considère des objets appartenant à une certaine classe. Mais quand un développeur applique un patron pour créer ses classes, un autre développeur pourra les reconnaître comme faisant partie de ce patron. Il s'agit bien d'un enrichissement du langage puisque les objets conçus à partir d'un patron auront une signification commune et supplémentaire (à condition, bien sur, de connaitre et de reconnaitre le patron). Cette sémantique est donnée dans le document (ou les connaissances) de référence qui décrivent le patron.
Ceci est d'autant plus vrai dans le domaine des méthodes formelles puisque la raison d'être des modèles (formels) est de porter une sémantique. En conception objet, les design patterns sont principalement vu comme un couple problème/solution. Il nous semble que, dans notre domaine, parler uniquement de problème est réducteur. Certes nous faisons face à un problème de modélisation mais il est plus juste de parler d'un aspect que l'on souhaite modéliser ou étudier.
En B évènementiel la solution apportée par le patron peut être résumée (du moins son encodage) par un modèle. Il s'agit de donner la forme générale et générique de la solution proposée pour exprimer l'aspect ou le comportement traité par le patron.

Cette expression doit être instanciée pour chaque occurrence d'un aspect identifié dans le système étudié. Le patron formalise un ensemble de concepts qui, ensemble, caractérisent un comportement particulier. Par exemple le comportement d'action-réaction est formé par les aspects d'action, de réaction forte et de réaction faible.

Mais ce processus ne résume que partiellement le processus qui l'on doit mettre en œuvre pour appliquer un patron. En effet, l'activité d'instantiation du patron sur le problème ne peut se faire que si l'on a bien identifié les comportements du système étudié. Le documentation de référence du patron doit aider à cette tâche en décrivant le contexte et les caractéristiques des aspects à identifier.

En B évènementiel, l'introduction d'un nouvel aspect dans un développement prouvé se fait lors d'un raffinement. On fait parfois la distinction entre un raffinement "horizontal" ou "vertical". Un raffinement horizontal superpose un nouveau comportement dans un modèle sans modifier ce qui existait déjà (il n'y a pas de preuve de raffinement à faire). Ce type de raffinement est utilisé pour introduire les spécifications en plusieurs étapes. Alors qu'un raffinement vertical transforme effectivement une donnée ou un évènement, c'est à dire que des éléments de la machine abstraite disparaissent (comme des variables, des gardes) et l'on doit montrer que les éléments que l'on a ajoutés simulent le même comportement. Dans ce type de raffinement, les obligations de preuves expriment les raffinements de données ou le passage de la spécification à l'implémentation.



Quand on considère un patron, les éléments que l'on ajoute lors du raffinement sont formalisés par le modèle du patron. On cherche donc à passer d'un modèle m0 à un modèle m1 en utilisant le patron P. Dans le cas d'une spécification initiale qui est directement temporelle, on peut prendre un modèle vide pour m0. Il est intéressant de noter que entre m1 et P nous avons aussi une relation de raffinement. Nous avons donc deux relations de raffinement, en pratique les outils logiciels ne sont pas capables de vérifier ce double raffinement. Dans la suite de ce chapitre nous montrerons effectivement la relation de raffinement entre le modèle de l'exemple m1 et le patron P. Mais en général, la relation entre m1 et P est facile à vérifier, il s'agit principalement de renommer les éléments et de les recopier plusieurs fois dans le modèle. C'est pourquoi nous ne détaillerons pas systématiquement et formellement la relation entre les modèles et le patron dans les études de cas. Par contre la relation entre m0 et m1 peut être complexe, car c'est ici que nous allons trouver les obligations de preuves introduisant les démonstrations sur le système étudié.

Il serait possible d'étendre les outils pour prendre en compte le raffinement multiple. Dans le cas général, cela peut être complexe à cause d'éléments partagés entre les modèles abstraits, par contre si tous ces modèles sont indépendants deux à deux (pas de variable, ni d'évènement partagés) alors il s'agit rien de plus qu'une relation de raffinement répétée. De cette manière il est possible d'importer des invariants qui serait définis au niveau du patron. Il n'y aurait plus besoin de les prouver directement par invariance mais de prouver que l'on raffine correctement le patron, c'est qui est souvent plus simple à réaliser.

Dans nos études de cas, il s'agit de montrer qu'un système temporel implémente correctement une spécification fonctionnelle (non-temporelle). Typiquement les premières machines et raffinements introduisent une spécification non-temporelle (raffinement horizontal), dans ce type de modèles le comportement temporel est exprimé abstraitement. Ensuite cette partie abstraite est supprimée au profit du vrai comportement temporel (raffinement vertical). Dans cette étape de raffinement, les obligations de preuves de raffinement permettent de démontrer la correction du comportement temporel.

3.2 Exemple récurrent

Pour illustrer nos propos, nous allons appliquer tous les patrons de ce chapitre sur le même exemple. Il s'agit d'un modèle simple mettant en jeu une lampe, du type de celle que l'on trouve dans les couloirs, associée à un minuteur qui se charge de l'éteindre après un certain laps de temps. L'état allumé de la lampe est exprimé par une variable de type booléen lo (*Light On*). Un évènement on représente l'allumage tandis qu'un évènement *off* représente l'extinction.

```
MACHINE m0
VARIABLES lo
INVARIANTS
  inv1: lo ∈ BOOL
EVENTS
INITIALISATIONS ≙ ...
on ≙ ...
off ≙ ...
END //m0
```

Initialement la lampe est éteinte.

```
INITIALISATIONS ≙
  BEGIN
    act1: lo := FALSE
  END
```

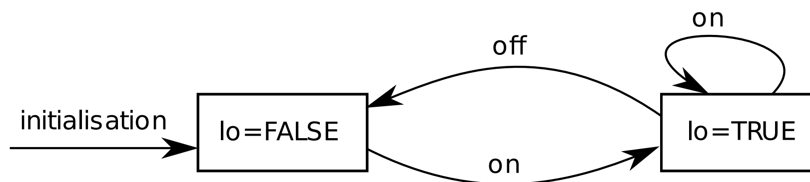
A n'importe quel moment on peut actionner le bouton qui allume la lampe (pas de garde).

```
on ≙
  BEGIN
    act1: lo := TRUE
  END
```

Si la lampe est allumée, le minuteur est susceptible de l'éteindre. Pour le moment nous n'avons pas de contraintes temporelles.

```
off ≙
  WHEN
    grd1: lo = TRUE
  THEN
    act1: lo := FALSE
  END
```

Les transitions possibles peuvent se voir dans le graphe ci-dessous.



Le modèle ci-dessus ne comporte pas d'argument temporel, il s'agit simplement d'un squelette sur lesquels nous allons appliquer nos patrons. Informellement, la contrainte temporelle à respecter est qu'il faut que l'évènement *off* survienne dans un délai compris entre $c - d$ et $c + d$ unité de temps après l'évènement on. Les constantes c et d étant définies dans un contexte avec comme axiome $c - d > 0$. La valeur d est là pour représenter l'imprécision possible du minuteur et c est la durée pendant laquelle on veut que la lampe soit allumée.

3.3 Encodage des fonctions totales par des variables

Dans les divers patron décrit dans ce chapitre nous utilisons beaucoup de fonctions totales ayant un domaine fini et de cardinalité constante et connue, par exemple

$$f \in \{a, b, c\} \rightarrow F.$$

En particulier, nous utilisons souvent un ensemble d'identifiants associé aux évènements d'un modèle (il n'est pas possible de faire directement référence à un nom d'évènement dans une modèle B évènementiel).

Ce type de fonctions peut être encodé par plusieurs variables (en fait autant que $\mathrm{card}(\mathrm{dom}(f))$). Il est ainsi possible de remplacer les valeurs provenant des applications de la fonction (par exemple $f(a), f(b), f(c)$) par ces variables (par exemple f_a, f_b, f_c). Les expressions plus complexes comme les quantifications peuvent aussi être remplacées en instanciant explicitement la quantification sur tout le domaine. L'intérêt principale de ce raffinement de donnée est de faire disparaitre ces quantifications, il est aussi plus simple de manipuler plusieurs variables qu'une seule fonction (mais moins concis).

3.4 Patron d'agenda absolu

Nous allons maintenant présenter un patron basé sur l'idée d'un agenda. Le but de ce patron est de fournir des dates de déclenchement futur pour certains évènements. C'est à dire que l'on va prévoir et forcer un évènement à se déclencher. Dans ce patron, le délai pour le déclenchement d'un évènement est explicitement considéré, il est spécifié dans une variable du modèle.

Nous l'avons nommé ce patron "agenda" puisqu'il sert à noter la date des évènements et nous l'avons qualifié d'absolu car l'instant zéro des dates est le début du système, c'est à dire l'instant où l'évènement d'initialisation se déclenche.

Modèle du patron

Le patron contient une variable at (*Activation Time*) qui est une fonction associant à chaque évènement (en fait à un identifiant qui représentera l'évènement) un ensemble de nombres. Cet ensemble de nombres sont les instants (dans le futur) où l'évènement associé sera déclenché. Nous avons de plus la variable now qui est le délai écoulé depuis le déclenchement de l'évènement d'initialisation (c'est à dire l'instant courant).

Nous trouvons ci-dessous les définitions et invariants du modèle du patron. Il faut de plus considérer un ensemble porteur F défini dans un contexte non représenté ici. Cet ensemble F définit des identifiants pour représenter les évènements du système sur lequel on applique le patron. Le dernier invariant trouvera son explication après la présentation du modèle.

```
MACHINE tp_at
VARIABLES now, at
INVARIANTS
  inv1: now ∈ ℕ
  inv2: at ∈ F → ℙ(ℕ)
  inv3: ∀e·e ∈ dom(at) ∧ at(e) ≠ ∅ ⇒ now ≤ min(at(e))
EVENTS
INITIALISATIONS ≙ ...
use ≙ ...
tic ≙ ...
END //tp_at
```

Initialement now se trouve donc logiquement à zéro (la valeur importe peu, on prend zéro par convention), quand aux dates d'activation contenues dans at elles sont libres.

```
INITIALISATIONS ≙
  BEGIN
    act1: now := 0
    act2: at :∈ F → ℙ(ℕ)
  END
```

L'évènement set du patron représente la modification des dates de l'agenda pour l'ensemble e d'identifiants d'évènements, il faut que les nouvelles dates soient dans le futur (après now).

set $\widehat{=}$
  ANY e,neat
  WHERE
    grd1: $e \subseteq dom(at)$
    grd2: $neat \in e \rightarrow \mathbb{P}(\{x|now < x\})$
  THEN
    act1: $at := at \Leftarrow neat$
  END

L'évènement use représente le déclenchement d'un évènement ayant pour identifiant e, il se déclenche quand la date courante (now) est égale à une date prévue dans l'agenda ($at(e)$) de cet évènement. Lors du déclenchement la date est enlevée de l'agenda.

use $\widehat{=}$
  ANY e
  WHERE
    grd1: $e \in dom(at)$
    grd2: $now \in at(e)$
  THEN
    act1: $at(e) := at(e) \setminus \{now\}$
  END

L'évènement tic de progression du temps fait avancer le temps courant now, toutes les dates de l'agenda sont des bornes supérieurs à cette progression du temps. En effet, nous voulons déclencher certains évènements à certaines dates, il faut donc faire des pas de progression du temps suffisamment petits pour ne pas "oublier" des évènements. En fait, forcer le déclenchement d'un évènement à une certaine date revient à bloquer le temps à cette date tant que cet évènement ne s'est pas déclenché. Plus précisément, la progression du temps est limitée à $\min(\operatorname{ran}(at))$. Lorsque l'évènement est effectivement déclenché (évènement use) la date est supprimée, ce qui débloque la progression du temps.

tic $\widehat{=}$
  ANY shift
  WHERE
    grd1: $0 < shift$
    grd2: $\forall e \cdot e \in dom(at) \land at(e) \neq \varnothing \Rightarrow now + shift \leq min(at(e))$
  THEN
    act1: $now := now + shift$
  END

Comme nous l'avons déjà dit, cette manière de faire implique que toutes les valeurs de l'agenda sont dans le futur, ce qui s'exprime dans l'invariant $inv3$ du modèle.

Il faut remarquer que ce patron propose un modèle assez fort (contraint) car en principe la présence d'un élément dans l'agenda d'un évènement est équivalent à la garde de cet évènement. L'exemple ci-dessous illustre cela.

Exemple

Nous retrouvons notre exemple de minuteur à lampe pour illustrer ce patron. On peut voir ci-dessous l'entête et les invariants. Nous avons effectivement indiqué deux raffinements dans ce modèle. Comme l'outil ne le gère pas, nous avons uniquement entré le raffinement entre le patron et le modèle. Pour vérifier manuellement le raffinement entre le modèle sans les contraintes temporelles et celui ci, il suffit de vérifier le raffinement de l'évènement *off*. Les invariants importants à considérer sont d'abord le numéro quatre qui précise que l'agenda a une cardinalité inférieure ou égale à un, et que l'unique date possible de l'agenda est entre le moment présent et $c + d$ unités de temps dans le futur. Quand à l'invariant numéro cinq il donné l'équivalence entre l'état de la lampe et le présence d'une date dans l'agenda. Il ne faut pas oublier que lors du raffinement on conserve les invariants abstraits, on a donc aussi le droit d'utiliser l'invariant $inv3$ du patron tp_at. Nous avons appliqué, le raffinement de donnés sur la fonction at ce qui permet d'utiliser uniquement la variable at_o à la place de $at(o)$, o étant l'identifiant associé à l'évènement on.

MACHINE m1_at
REFINES m0,tp_at
VARIABLES lo, now, at_o
INVARIANTS
inv1: $lo \in BOOL$
inv2: $now \in \mathbb{N}$
inv3: $at_o \subseteq \mathbb{N}$
inv6: $at = \{o \mapsto at_o\}$
inv4: $\exists x \cdot x \in now \mathinner{..} now + c + d \wedge at_o \subseteq \{x\}$
inv5: $lo = FALSE \Leftrightarrow at_o = \varnothing$
EVENTS
INITIALISATIONS $\widehat{=}$ ...
on $\widehat{=}$ ...
off $\widehat{=}$ ...
tic $\widehat{=}$ ...
END //m1_at

L'initialisation est prévisible avec la lampe éteinte, un temps courant égale à zéro et pas de valeurs dans l'agenda.

INITIALISATIONS $\widehat{=}$
WITH
at': $at' = \{o \mapsto at_o'\}$
BEGIN
act1: $lo := FALSE$
act2: $now := 0$
act3: $at_o := \varnothing$
END

Quand la lampe s'allume, on prend une date (de manière non déterministe) dans l'intervalle possible et on l'ajoute à l'agenda. Il faut remarquer que dans ce cas, cette valeur est connue et peut être utilisée dans les preuves.

on $\widehat{=}$
REFINES on,set
ANY dc
WHERE
grd1: $dc \in c - d \mathinner{..} c + d$
WITH
neat: $neat = \{o \mapsto \{now + dc\}\}$
e: $e = \{o\}$
THEN
act1: $lo := TRUE$
act2: $at_o := \{now + dc\}$
END

L'arrêt de la lumière (évènement *off*) se déclenche quand l'instant courant est égal à la date placée dans l'agenda at_o. C'est un modèle assez fort car on est capable de distinguer les cas ou l'évènement *off* est en attente de déclenchement (lampe allumée) avec la valeur précise de ce déclenchement et le cas ou la lampe est éteinte. Il y a en fait une équivalence entre la valeur de lo et celle de l'agenda (voir invariant *inv5*). Il s'ensuit qu'on peut complètement remplacer la garde abstraite par la présence de l'instant présent dans l'agenda (raffinement de l'évènement off avec disparition d'une garde).

off $\widehat{=}$
REFINES off,use
WHEN
grd2: $now \in at_o$
WITH
e: $e = o$
THEN
act1: $lo := FALSE$
act2: $at_o := at_o \setminus \{now\}$
END

Dans ce patron, l'évènement tic du progression du temps est complètement conforme à celui du patron et nous n'avons pas besoin d'apporter des modifications (en dehors du raffinement de donné de la fonction at).

tic $\widehat{=}$
REFINES tic
ANY shift
WHERE
grd2: $0 < shift$
grd3: $at_o \neq \varnothing \Rightarrow now + shift \leq min(at_o)$
THEN
act1: $now := now + shift$
END

3.5 Patron d'agenda relatif

Le patron d'agenda utilise une variable (now) pour spécifier le temps courant du système, ceci permet à l'évènement tic d'avoir une forme très simple : il suffit d'incrémenter now pour représenter le passage du temps. Cette variable augmente donc indéfiniment et, en cas d'utilisation d'un *model-checker*, cela pose problème puisque le nombre d'état est infini à cause de cette variable. Pour éviter d'introduire systématiquement des états différents à cause de now, on peut supprimer cette variable du modèle et remplacer l'agenda absolu at par un agenda relatif rat (*Relative Activation Time*). Nous l'avons qualifié de relatif car les dates contenu dans l'agenda utilisent le temps courant comme repère plutôt que l'initialisation du système.

Cette verson de l'agenda est équivalente à la version absolue, c'est à dire qu'en considérant la formule

$$\forall e \cdot e \in dom(at) \Rightarrow (\forall x \cdot x \in rat(e) \Leftrightarrow x + now \in at(e))$$

on peut réécrire les patrons de l'un vers l'autre et vice-versa. Nous avons d'ailleur prouvé le raffinement dans les deux sens entre les deux versions.

Ces nouvelles valeurs peuvent être vus comme des comptes-à-rebours avant le déclenchement des évènements associés. Comme toutes les valeurs ont pour repère le temps courant now, cette variable n'est plus utile : le temps courant est simplement zéro. L'évènement tic de passage du temps s'en trouve modifié, il faut en effet faire décroitre toutes les valeurs de rat pour représenter le passage du temps.

Modèle du Patron Le modèle du patron est similaire à celui de l'agenda, avec évidement les conséquences de la modification expliquée ci-dessus. La variable now disparait et est remplacée par 0 si nécessaire. L'agenda at est replacé par l'agenda relatif rat. Au niveau de l'invariant, il suffit maintenant de prendre le co-domaine de rat dans les entiers positifs pour s'assurer que les date de l'agenda sont dans le futur. Et similairement on retrouve les trois évènements qui composaient le modèle du patron d'agenda absolu.

MACHINE tp_rat
SEE tp_c0
VARIABLES rat
INVARIANTS
inv1: $rat \in E \rightarrow \mathbb{P}(\mathbb{N})$
EVENTS
INITIALISATIONS $\widehat{=} \ldots$
set $\widehat{=} \ldots$
use $\widehat{=} \ldots$
tic $\widehat{=} \ldots$
END //tp_rat

L'initialisation des valeurs est libre.

INITIALISATIONS $\widehat{=}$
BEGIN
act1: $rat :\in E \rightarrow \mathbb{P}(\mathbb{N})$
END

Pour ajouter, modifier ou remplacer des dates de l'agenda sur un ensemble e d'identifiant d'évènements on dispose des nouvelles valeurs $nerta$ qui sont surchargées sur l'agenda.

set $\widehat{=}$
ANY e,nerat
WHERE
grd1: $e \subseteq dom(rat)$
grd2: $nerat \in e \rightarrow \mathbb{P}(\mathbb{N}_1)$
THEN
act1: $rat := rat \Leftarrow nerat$
END

Quand le compte-à-rebours d'une date de l'agenda arrive à zéro on peut déclencher l'évènement associé et supprimer cette valeur de l'agenda.

use $\widehat{=}$
ANY e
WHERE
grd1: $e \in dom(rat)$
grd2: $0 \in rat(e)$
THEN
act1: $rat(e) := rat(e) \setminus \{0\}$
END

Le passage du temps a pour effet de décroitre les valeurs de l'agenda. Pour exprimer cela on utilise la fonction $adds$ qui prend en entier en paramètre et renvoie une fonction incrémentant toutes les valeurs d'un ensemble par cet entier, par exemple $adds(2)(\{1,3\}) = \{3,5\}$ (voir la définition ci-après).

tic $\widehat{=}$
ANY shift
WHERE
grd1: $0 < shift$
grd2: $\forall e \cdot e \in dom(rat) \wedge rat(e) \neq \varnothing \Rightarrow shift \leq min(rat(e))$
THEN
act1: $rat := rat; adds(-shift)$
END

La fonction $adds$ est définie par les axiomes suivants:

CONSTANTS adds
AXIOMS
axm4: $adds \in \mathbb{Z} \rightarrow (\mathbb{P}(\mathbb{Z}) \rightarrow \mathbb{P}(\mathbb{Z}))$
axm5: $\forall as, bs, c \cdot as \subseteq \mathbb{Z} \wedge bs \subseteq \mathbb{Z} \wedge c \in \mathbb{Z}$
$\Rightarrow (as \mapsto bs \in adds(c) \Leftrightarrow (\forall a \cdot a \in as \Leftrightarrow a + c \in bs))$

3.6 Exemple

Notre exemple courant peut ainsi être réécrit de la manière exposée ci-dessous. On peut vérifier que les propriétés en invariants sont les même modulo la réécriture entre at et rat. Et ce modèle possède un nombre d'état fini, à condition, bien sur, de donner une valeur aux constantes c et d.

MACHINE m1_rat
REFINES tp_rat
SEE c1, c1_at
VARIABLES lo,rat_o
INVARIANTS
inv1: $lo \in BOOL$
inv2: $rat_o \in \mathbb{P}(\mathbb{N})$
inv5: $rat = \{o \mapsto rat_o\}$
inv3: $\exists x \cdot x \in 0 \mathinner{..} c + d \wedge rat_o \subseteq \{x\}$
inv4: $lo = FALSE \Leftrightarrow rat_o = \varnothing$
EVENTS
INITIALISATIONS $\widehat{=} \dots$
on $\widehat{=} \dots$
off $\widehat{=} \dots$
tic $\widehat{=} \dots$
END //m1_rat

INITIALISATIONS $\widehat{=}$
WITH
rat': $rat' = \{o \mapsto rat_o'\}$
BEGIN
act1: $lo := FALSE$
act3: $rat_o := \varnothing$
END

on $\widehat{=}$
REFINES on,set
ANY dc
WHERE
grd1: $dc \in c - d \mathinner{..} c + d$
WITH
e: $e = \{o\}$
nerat: $nerat = \{o \mapsto \{dc\}\}$
THEN
act1: $lo := TRUE$
act2: $rat_o := \{dc\}$
END

off $\widehat{=}$
REFINES off,use
WHEN
grd1: $0 \in rat_o$
WITH
e: $e = o$
THEN
act1: $lo := FALSE$
act2: $rat_o := rat_o \setminus \{0\}$
END

De nouveau l'évènement tic est simplifié par le raffinement de donné sur la fonction rat. Ce qui permet d'utiliser une notation avec un ensemble par compréhension plus simple que la définition de $adds$ pour décrémenter les valeurs.

tic $\widehat{=}$
  REFINES tic
  ANY shift
  WHERE
    grd1: $0 < shift$
    grd3: $rat_o \neq \varnothing \Rightarrow shift \leq min(rat_o)$
  THEN
    act1: $rat_o := \{x \cdot x \in rat_o | x - shift\}$
  END

3.7 Patron de chronomètres

Ce patron de chronomètre est le dernier que nous avons développé. Il provient d'une recherche pour dégager l'élément fondamental nécessaire à l'étude des propriétés temporelles quantitatives. Cet élément est la durée qui s'est écoulée depuis le dernier déclenchement d'un évènement. Par analogie, nous allons appeler cette donnée le chronomètre d'un évènement. Elle est a des similarités avec les horloges des automates temporisés. Mais nous avons choisi le terme chronomètre car cette donné est remise à zéro lors du déclenchement de l'évènement surveillé, alors qu'une horloge n'est à priori jamais remise à zéro.

Le patron comporte plusieurs volets, le premier est la machine B évènementielle contenant le modèle du temps que nous allons par la suite raffiner et introduire dans les modèles à étudier.

3.7.1 Modèle du patron

Nous avons en effet besoin d'un modèle qui représentera le phénomène du temps et de son écoulement. Plus précisément, nous allons considérer un ensemble de valeurs nommées s'incrémentant de manière uniforme avec le passage du temps et pouvant être remis à zéro individuellement. Pour cela prenons un modèle tp_s (*Time Pattern Since*) présentant une variable s (*Since*) qui est une fonction totale d'un ensemble E d'identifiant d'évènement vers les entiers.

MACHINE tp_s
VARIABLES s
INVARIANTS
  inv1: $s \in E \rightarrow \mathbb{N}$
EVENTS
INITIALISATIONS $\widehat{=} \ldots$
reset $\widehat{=} \ldots$
tic $\widehat{=} \ldots$
END //tp_s

Le modèle possède trois évènements que nous détaillons çi-dessous. Le premier est l'initialisation.

INITIALISATIONS $\widehat{=}$
  BEGIN
    act1: $s :\in E \rightarrow \mathbb{N}$
  END

On remarquera que la valeur initiale est prise sans contrainte dans l'ensemble des entiers positifs. Nous avons ensuite l'évènement représentant la remise à zéro d'un chronomètre particulier.

reset $\widehat{=}$
  ANY e
  WHERE
    grd1: $e \in E$
  THEN
    act1: $s(e) := 0$
  END

Et enfin, l'évènement représentant la progression du temps.

tic $\widehat{=}$
  ANY *shift*
  WHERE
    grd1: 0 < *shift*
  THEN
    act1: $s := \{e \cdot e \in E | e \mapsto s(e) + shift\}$
  END

Cet évènement incrémente la valeur de tous les chronomètres par une valeur *shift* non nulle.

3.7.2 Insertion du patron raffiné

Le modèle du patron ne constitue que partiellement le patron. En effet, ce modèle va être utilisé d'une manière précise et cette méthode d'utilisation fait aussi partie du patron. Comme nous l'avons déjà dit, l'idée de base est de considérer la durée s'étant écoulée depuis le dernier déclenchement d'un évènement. Pour cela, nous allons superposer au modèle à étudier une version raffinée du patron. Pour tout évènement chronométré on va lui superposer l'évènement *reset* du patron en raffinant la variable e par un identifiant représentant l'évènement.
Exemple : On prend comme modèle à étudier l'exemple de la lampe de ce chapitre. Le seul évènement que l'on veut chronométrer est on, le résultat de l'application du patron est donné ci-dessous.

on $\widehat{=}$
  REFINES on,reset
  WITH
    e: $e = o$
  BEGIN
    act1: $lo := TRUE$
    act2: $s(o) := 0$
  END

Il faut répéter cette opération de superposition pour chaque évènement que l'on veut chronométrer. Ainsi l'ensemble E doit contenir des identifiants en bijection avec les évènements étudiés.
Enfin il convient d'insérer tel quel l'évènement tic du patron.
Nous avons ainsi superposé dans le modèle à étudier les éléments nécessaires pour représenter des contraintes et des propriétés temporelles (que nous ajouterons par la suite). Mais auparavant, nous pouvons apporter une simplification sous la forme d'un raffinement de données.
En effet, une variable fonction totale (par exemple $f \in E \rightarrow F$) ayant pour domaine un ensemble fini, de cardinalité n constante et connue peut être remplacée par n variables (distinctes). Il suffit pour cela de remplacer les termes de la forme $f(x)$, avec $x \in E$, par un variable que l'on appellera par convention f_x (qui doit évidement ne pas déjà être une variable libre). De plus cela permet de simplifier l'expression de l'incrémentation dans l'évènement tic.
Dans notre exemple les deux évènements deviennent :

on $\widehat{=}$
  REFINES on,reset
  WITH
    e: $e = o$
  BEGIN
    act1: $lo := TRUE$
    act2: $s_o := 0$
  END

tic $\widehat{=}$
  REFINES tic
  ANY shift
  WHERE
    grd1: $0 < shift$
  THEN
    act1: $s_o := s_o + shift$
  END

Il faut aussi ajouter l'invariant de typage et de collage

inv2: $s_o \in \mathbb{N}$
inv5: $s = \{o \mapsto s_o\}$

Remarquons que si la cardinalité de E est n on aurait n lignes similaires à $act1$ dans l'évènement tic.
Nous faisons systématiquement ce raffinement après l'application du patron car un ensemble fini de variables est plus simple à manipuler qu'une fonction, en particulier au niveau de l'évènement tic (les preuves associées à tic peuvent être nombreuses). Notons que ce raffinement de donné ne transforme pas le comportement du système.
De manière plus anecdotique, le paramètre $shift$ de l'évènement de progression du temps peut être raffiné par la constante 1.
Ceci clos le volet de l'insertion du patron, nous avons inséré dans le modèle un modèle du temps mais nous n'avons pas encore ajouté de réel comportement temporel au système, nous n'avons pour le moment que la structure pour le faire.

3.7.3 Représentation des contraintes temporelles

Nous allons ajouter des contraintes sur la durée qui doit s'écouler entre deux déclenchements d'évènements. Et ceci est possible entre deux évènements différents ou pour le même évènement cela ne change rien à la méthode.

Borne supérieure bloquante

Considérons un premier cas (le plus compliqué) : nous voulons forcer la durée d entre deux évènements e et f à être inférieur à une certaine valeur v. Notons bien qu'il s'agit d'une obligation pour le déclenchement de l'évènement f et non pas d'une permission. L'évènement e n'est pas impacté, il sert juste de repère temporel et l'on doit disposer d'un chronomètre s_e (*Since E*) sur cet évènement. Dans ce cas là, il s'agit d'une borne supérieur sur la durée d. Pour représenter cette borne supérieur dans le modèle, nous allons placer une contrainte sur la progression du temps (c'est à dire l'évènement *tic*). L'idée est que dire qu'un évènement doit se produire avant un moment donné, revient à dire que le temps ne doit pas s'écouler au delà de ce moment tant que l'évènement n'a pas été exécuté.
Pour bloquer la progression du temps il faut placer une borne supérieure sur la valeur *shift*. Rappelons que l'évènement tic de progression du temps incrémente tous les chronomètres par une valeur *shift* non nulle. Si on veut limiter le delai d à une valeur a, il faut alors que $\mathit{shift} \leq a - d$, de manière équivalente $d + \mathit{shift} \leq a$. En fait le délai d est donné par le chronomètre s_e .
Mais cette limitation de *shift* ne doit intervenir que lorsque l'évènement f peut se déclencher ; c'est à dire lorsque sa garde G est vrai. Finalement nous devons donc ajouter en tant que garde de *tic* le prédicat

$$G \Rightarrow s_e + \mathit{shift} \leq a.$$

Exemple : Dans notre exemple de lampe, nous voulons que la lampe soit éteinte après un délai dans l'intervalle $c - d \mathrel{..} c + d$. Pour le moment, nous allons seulement considérer l'obligation pour la lampe d'être éteinte après $c + d$ unités de temps. La garde de l'évènement *off* qui éteint la lampe est $lo = TRUE$ et le chronomètre de l'évènement *on* qui l'allume est s_o ; il faut donc ajouter la garde $grd2$ à tic, voir ci-dessous.

tic $\widehat{=}$
REFINES tic
ANY shift
WHERE
grd1: $0 < shift$
grd2: $lo = TRUE \Rightarrow s_o + shift \leq c + d$
THEN
act1: $s_o := s_o + shift$
END

Il est possible que nous ayons besoin de considérer non pas un chronomètre par évènement mais un ensemble de chronomètres par évènement. Par exemple dans le cas d'un système distribué si l'évènement représente une action que réalise tout un ensemble d'appareils distribués alors il faut bien considérer un chronomètre pour chaque système en particulier. Formellement on peut donc associe le chronomètre à l'évènement paramétré par une variable qui représente l'entité distribuée.

Borne inférieure

Notre deuxième cas est un borne inférieure b sur le délai entre le déclenchement d'un évènement e et f. La solution est simple, il suffit d'ajouter une garde $b \leq s_e$ dans f.
Exemple : La lampe peut s'éteindre à partir de $c - d$ unités de temps, on ajoute donc la garde $grd2$ dans *off*.

```
off ≙
  WHEN
    grd1: lo = TRUE
    grd2: c − d ≤ s_o
  THEN
    act1: lo := FALSE
  END
```

3.7.4 Représentation des propriétés temporelles

Nous avons maintenant définit comment obtenir une machine contenant un modèle du temps et des contraintes temporelles. Le dernier volet de l'application du patron consiste à écrire les propriétés du système que l'on veut vérifier sous la forme d'un invariant. Pour cela nous pouvons utiliser les valeurs des chronomètres pour décrire les propriétés que nous voulons. Celles-çi dépendent du système, et l'on peut utiliser toutes expressions arithmétiques nécessaires légales en B évènementiel pour les exprimer.
On peut aussi remarquer que les bornes supérieures des contraintes temporelles, celles qui apparaissent dans la garde de tic, mènent systématiquement à une clause d'invariant. En effet, en considérant un chronomètre s_e (d'un évènement e avec une garde G) avec une borne supérieure a, nous avons systématiquement en invariant :

$$G \Rightarrow s_e \leq a$$

Cela se vérifie aisément au vue de la forme des gardes de *tic*.
Exemple : Dans l'exemple de la lampe, nous trouvons d'abord l'invariant tel que décrit ci dessous puis un autre concernant la borne inférieure.

```
inv3: lo = TRUE ⇒ s_o ≤ c + d
inv4: lo = FALSE ⇒ c − d ≤ s_o
```

Il faut noter qu'un invariant comme inv4 n'est pas systématiquement valide.
De plus, lors de la conception du modèle il faut prendre garde à l'initialisation. En effet le concept des chronomètres sous-entend que dans tous les états du système, tous les évènements se sont déclenchés dans le passé et on connait la durée depuis laquelle cela s'est produit. Évidement ce n'est pas le cas lors de l'initialisation, il faut donc donner une valeur artificielle aux chronomètres de manière à respecter l'invariant. Si on veut éviter d'avoir à considérer ce cas d'initialisation, il est possible d'utilise un type plus complexe que les entiers ou l'on pourrait avoir une valeur spéciale indiquant que l'évènement ne s'est jamais déclenché dans le passé. Toutefois le recours à ce type d'expression complexifie le modèle et ne justifie pas systématiquement. De plus rappelons que l'initialisation est surtout là pour démontrer qu'il existe un état initial qui respecte l'invariant.
Exemple : pour l'initialisation de l'exemple, comme le système commence avec la lampe éteinte il faut prendre une valeur pour le chronomètre après la borne supérieur du délai d'extinction.

```
INITIALISATIONS ≙
  WITH
    s': s' = {o ↦ c + d + 1}
  BEGIN
    act1: lo := FALSE
    act2: s_o := c + d + 1
  END
```

3.8 Conclusion

Dans ce chapitre nous avons définit notre point de vue sur les patrons et donné en exemple trois patrons servant à aider à la modélisation de systèmes temporels. Nous pensons que ces exemples montrent l'intérêt de capitaliser l'expérience obtenue en modélisant et prouvant des systèmes. Ces patrons sont à prendre

comme le fruit d'expérimentations sur des études de cas et nous espérons qu'ils permettront à d'autre personnes d'accélérer leurs études de systèmes temporisés.
Les deux concepts (en regroupant les variantes de l'agenda relatif et absolu) que nous avons défini sont complémentaires, en effet, le patron d'agenda permet de contraindre fortement le déroulement d'un système dans le temps tandis que le patron de chronomètre superpose des contraintes temporelles de manière plus lâche sur le comportement sur système.

Chapter 4

Applying patterns for modelling pacemaker-like systems

Sommaire

4.1 Overview of Pacemaker System and Environment **44**
4.1.1 The Heart Environment 44
4.1.2 The Pacemaker 44
4.2 Event-B Patterns **45**
4.2.1 Action-Reaction Pattern 46
4.2.2 Time-Based Pattern 46
4.3 Overview of Pacemaker System Modelling **47**
4.4 Abstract model of Pacemaker **49**
4.4.1 Abstraction of AOO and VOO modes 49
4.4.2 Abstraction of AAI and VVI modes 51
4.4.3 Abstraction of AAT and VVT modes 52
4.5 First refinement **53**
4.6 Second refinement:Threshold **53**
4.7 Third refinement:Hysteresis **55**
4.8 Fourth refinement:Rate Modulation **57**
4.9 Model Validation using ProB **59**
4.10 Conclusion **59**

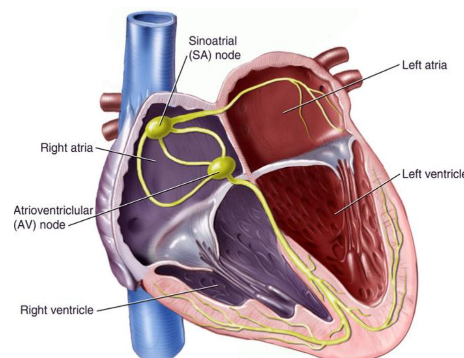
Ce chapitre a été rédigé par Neera Singh et Dominique Méry.

4.1 Overview of Pacemaker System and Environment

In this modelling, the pacemaker system consider as an embedded system operating inside the heart for controling the heart rate. First of all we review the basic elements of the heart environment that interact with pacemaker and then described the elements of pacemaker system itself.

4.1.1 The Heart Environment

The human heart is wondrous in its ability to pump for the circulatory system continuously throughout a lifetime. The heart's mechanical system (the pump) requires at the very least impulses from the electrical system. The heart consists of four compartments:the right and left atria and ventricles, which contract and relax periodically under the control of natural electrical stimuli. The atria form one unit and the ventricles another. The left ventricular free wall and the septum are much thicker than the right ventricular wall. This is logical since the left ventricle pumps blood to the systemic circulation, where the pressure is considerably higher than for the pulmonary circulation, which arises from right ventricular outflow. In the normal functioning of natural pacemaker or heart, a discharge is made at the sinus node; the discharge subsequently reaches the atrioventricular (AV) node which amplifies it, stimulating the ventricles. If the natural pacemaker is malfunctioning, a physical condition termed Bradycardia may arise in which the heart rate falls below the level expected for the patient [86]. To normalize the heart rate, an artificial pacemaker may be implanted to help the heart. The bpm (beats per minute) is a basic unit to measure the rate of the heart activity.



Heart or Natural Pacemaker

4.1.2 The Pacemaker

A pacemaker is an electronic device implanted in the body to regulate the heart beat. The pacmaker system composed of :

Leads: One or more flexible coiled metal wire,normally two, that transmits electrical signals between the heart and pacemaker. Each pacemaker lead is classified by whether it is configured with one ("unipolar") or two ("bipolar") separate points of electrical contact within the heart.

The Pacemaker Generator: The pacemaker is both the power source and the brains of the pacing system. As such, it contains a implanted batteries and controller as an electronic circuitry.

Device Controller-Monitor (DCM): An external unit that interacts with the pacemaker device using a wireless connection.

Accelerometer: It is a specific unit inside the pacemaker for measuring body motion in order to allow modulated pacing.

Table 4.1: Bradycardia operating modes of pacemaker system

Category	Chambers Paced	Chabers Sensed	Response to Sensing	Rate Modulation
Letters	**O**-None **A**-Atrium **V**-Ventricle **D**-Dual(A+V)	**O**-None **A**-Atrium **V**-Ventricle **D**-Dual(A+V)	**O**-None **T**-Triggered **I**-Inhibited **D**-Dual(T+I)	**R**-Rate Modulation

Artificial Pacemaker

In the single electrode pacemaker, the electrode attached with right atrium or right ventricle. In single electrode pacemaker has several operational modes that control the malfunctions of the heart. The specifications document [68] described all possible operating modes for controling the different parameters of the pacemaker. Most of the parameters related with real-time and action-reaction constraint for controling the interval between a pace in the atrium and the ventricle or the number of pulses per minute the device should deliver to a given chamber.
Pacemaker function is described by a universally accepted code consiststing of three or four characters. The code provides for a description of pacemaker pacing and sensing function using a four-letter sequence. It is the sequence, that is referred to as the "pacemaker mode". In practice, only the first three or four-letter positions are commonly used to describe bradycardia pacing function (i. e. , "AOO" or "VVIR"). The first letter of code sequence of operating mode represents that chamber paced ("O" for none, "A" for atrium, "V" for ventricle, "D" for both), second letter of code sequence of operating mode represents that chamber sensed ("O" for none, "A" for atrium, "V" for ventricle, "D" for both), third letter of code sequence of operating mode represents that response to sensing ("O" for none, "I" for inhibits pacing, "T" for triggers pacing, "D" for both inhibits and triggers pacing) and final optional letter of code sequence of the operating mode indicates the presence of rate modulation in response to the physical activity of the patient as measured by the accelerometer. "X" is a wildcard used to denote any letter (i. e. "O", "A", "V" or "D"). $Triggered$ refers to pacing in the chamber paced after the sensing of intrinsic activity in the chamber sensed. The sensing and pacing may occur in different chambers.

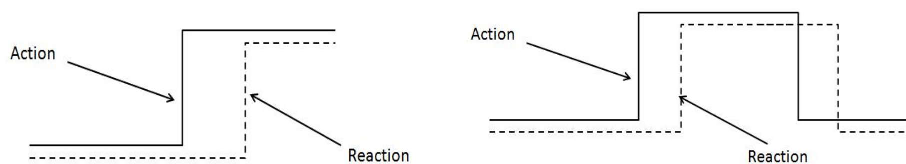
4.2 Event-B Patterns

The purpose of a design pattern is to capture structures and decisions within a design that are common to similar modeling and analysis tasks. They can be re-applied when undertaking similar tasks to in order reduce the duplication of effort. The pattern approach is the possibility to reuse solutions from former developments in the current project, means a new refinement of the problem at hand where a certain part of the model is replaced accordingly to a pattern that already exists. This will lead to a correct refinement in the chain of models in the development, without arising proof obligations. Since the correctness of the pattern has been proved during its development, nothing is to prove again when using this pattern [7].
Pacemaker systems are characterized by the fact that their behavior is defined in terms of action-reaction and real time patterns. Sequences of inputs are recognized, and outputs can be emitted in response within

a fixed time interval. So, the most basic elements in pacemaker systems are action, reaction and bounded time interval for every action, reaction and action-reaction pairs. The action-reaction within in a time limit can be viewed as an abstraction of the pacemaker system. We recognized the following two design patterns when modeling this kind of systems according to the relationship between the action and corresponding reaction.

4.2.1 Action-Reaction Pattern

In the action-reaction pattern we have five type of design patterns as follows:-
Action and Weak Reaction: When an action is emitted, a reaction should start in response to the action. If the action stops to stimulate sequentially, the reaction should also follow it to stop. However, the reaction sometimes has not enough time to react, because the action moves too quickly. This is so-called action and weak reaction.
Action and Strong Reaction: In some conditions, we hope that the every reaction should always follow the every actions. The action and reaction can always keep proper synchronization then this behaviour of action-reaction is known as pattern of action and strong reaction.
Composit Weak and Strong Reactions: Action and Reaction (weak or strong) are only the basic blocks for modeling discrete event system. In most cases, system to be modeled has some complex situations to handle, because functions of a large complex system depend on some sequences of events, in which some events may be of action-reaction relation and some may occur simultaneously. The interaction between two action-reaction blocks can be modeled as composite or synchronization, which depend on that the two blocks are of weak-strong reactions or strong-strong reactions. When the weak reaction of a specific action-reaction block results eventually in the specific strong reaction of some action-reaction, it can be recognized as the composite for weak and strong reactions.
Weak synchronization of two strong reactions: As far as the synchronization of two strong action-reaction blocks is concerned, two kinds of synchronizations could be identified, which can be recognized as weak synchronization and strong synchronization. The second strong reaction can be set in on state when the first strong reaction already in on state, but there is not any constraint for how many times the first strong reaction is set to on state and what will be state of first strong reaction after the off state of second strong reaction. This is what we called weak synchronization of two strong reactions.
Strong synchronization of two Strong reactions: Another kind of synchronization between two strong action-reaction blocks is so-called strong synchronization of two strong reactions. In this pattern given the solution of the problem with the weak synchronization of two strong reactions. The strong synchronization between two strong action-reaction blocks really means that the second reaction will strictly run after the first reaction , which reacts to the first action a and changes its value into on or off regularly.
Above action-reaction patterns are the refinements of weak action-reaction patterns [2].



Action-Reaction Patterns

4.2.2 Time-Based Pattern

The pacemaker system is highly based on time constraints pattern. All the action-reaction activities of electrodes of pacemaker based on hard real-time constraint. The time constraints pattern introduced in Event-B modelling by D. Cansell,D. Mery and Joris Rhem and successfully applied on IEEE 1394 case study [70]. We have also applied same time patterns to solve the time constraints of pacemaker system. This time pattern is fully based on timed automata. Timed automata are especially useful to model components of real-time systems. A timed automaton is a finite state machine extended with clock variables which evaluate to a nonnegative real number. The timed automata in a model and the way they interact with each other together define a timed transition system. Besides ordinary action transitions that can represent input, output and internal actions, a timed transition system has time passage transitions. Such time passage transitions result in synchronous progress of all clock variables in the model. This makes it possible to base

decisions on the moment in time on which they are taken, which is done by so-called clock constraints. A clock constraint depends on one or more clock values and can either be an invariant of a component state (defining the time period(s) in which the component may reside in this state) or a guard on a transition (defining under which time conditions this transition may be taken). Here we used the time pattern in modelling to synchronize the sensing and pacing behaviours of pacemaker in continuous progressive time constraint. All the events in the system guard with the time constraint means action of any event will be activated only when time constraints will satisfy at specific time. The time progress is also an event, so there is no modification of the underlying language of B. It is only a modelling technique instead of a specialised formal system. The variable $time$ is in $\mathbb{N}$ but time constraints can be written in terms involving unknown constants or expressions between different times. Finally, the timed event observations can be constrained by other events which determine future activations [70].

4.3 Overview of Pacemaker System Modelling

The pacemaker system is defined by a proof-based development of Event-B models which are modelling techniques in a very abstract and general way. In this study, we try to model all the modes of single electrode pacemaker system. We are applying the action-reaction and real-time patterns to model the single electrode pacemaker system. Each mode of pacemaker has specific properties to control the rate of natural heart. In order to understand the basic timing of a pacemaker one must understand the terminology commonly used to describe the events that occur. All single chamber pacemakers have three basic timed events:-

Automatic Interval: The period of time between two sequential paced beats uninterrupted by a sensed beat. It is also referred to as the base pacing interval and may be converted to bpm and expressed as the base pacing rate.

Escape Interval: The period of time after a sensed event until the next paced event. The escape interval is usually the same as the automatic interval. It may be different if a feature called "hysteresis" is enabled.

Refractory Period: This is a period of time after a paced or sensed event during which the pacemaker sensing is disabled. An event occurring during a refractory period will not be sensed, or will be "tagged" by the pacemaker as a refractory sensed event and used by the device for evaluation of possible abnormal rhythms (e. g. , atrial fibrillation). The reason for having a refractory period in a ventricular pacemaker is to prevent sensing of the evoked QRS and T-wave that occurs immediately after the paced event. In atrial pacemakers the refractory period also prevents sensing of the far-field R-wave or T-wave. In some devices the first part of the refractory period may be an adjustable "Blanking Period", during which no sensing at all occurs, followed by the remainder of the refractory period during which sensing occurs for diagnostic purposes only [77, 84, 62].

To model the all modes of single electrode pacemaker, we try to find the patterns, relation between all pacing modes and common pacing and sensing behaviours in all modes. We are applying the stepwise refinements to model all modes. Here we will just present a sufficient overview of the abstract specification and refinement stages in order to help the reader understand the basic notion of each refinement.

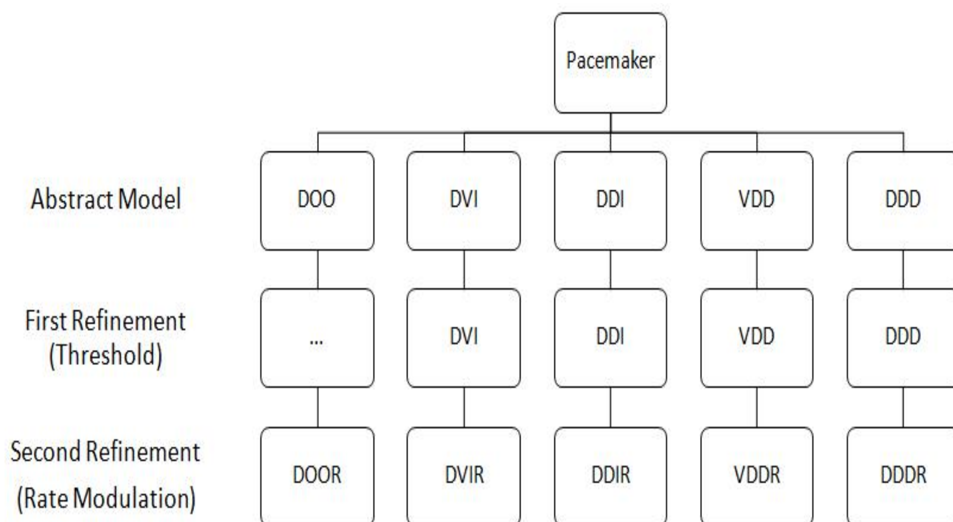
Abstract Model: In the abstract model of stepwise development of single electrode pacemaker contains the definition and properties of different time interval parameters (URL (Upper Rate Limit), LRL (Lower Rate Limit),. . . etc.) and pacemaker actuator status (ON and OFF). The first model contains the four basic events $Pace_ON$, $Pace_OFF$, tic and Set_Pace_Int, which are elementary events of single chamber pacing modes(AOO,VOO). Two extra new events $Pace_OFF_with_Sensor$ and $Sense_ON$ introduced in single chamber pacing modes(AAI,VVI). Similarly two more events $Pace_ON_with_Sensor$ and $Sense_ON$ introduced in single chamber pacing modes(AAT,VVT). Remaining other modes of single electrode pacemaker (AOOR, VOOR, AAIR,AATR,VVIR and VVTR) are refinement of basic single chamber pacing modes, which are describing in following continuous refinements. In the basic abstract model of pacemaker we introduced the action-reaction and real-time pattern for describing the pacing and sensing mode of single electrode pacemaker.

First Refinement: In the first refinement of the model we introduced the only some extra invariants in the abstract model to stable the system and make more strong for proper pacing and sensing at specific time constraint.

Second Refinement: This refinement is relatively more complex then the last refinement in which we introduced the threshold variable. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of a heart and a pulse generator for deliverying stimulation pulses to

the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value and a safety margin. The new event Thr_value introduced to take the value of $threshold$ variable.

Third Refinement: In this refinement, we have introduced the concept of $Hysteresis$ in pacing and sensing mode of single electrode pacemaker. Hysteresis, from the Greek for "to lag behind," means a delay of effect behind the cause. In pacemakers, this means delaying pacing to maximize patient benefit. The application of a hysteresis interval to provide consistent pacing of the atrial or ventricle, or to prevent constant pacing of the atrial or ventricle.



Refinement Structure of Pacemaker Operating Modes

Fourth Refinement: It is the final and last refinement of the single electrode pacemaker system. In this refinements we introduced the rate adapting pacing technique to the pacemaker. This refinement of pacemaker also give some new pacing and sensing mode (AAIR,VVIR,. . . etc.) of the pacemaker. The rate adapting mode of pacemaker can progressively pace faster than the lower rate, but no more than the upper sensor rate limit, when it determines that heart rate needs to increase. This typically occurs with exercise in patients that cannot increase their own heart rate. The amount of rate increase is determined by how much exertion, the pacemaker thinks the patient is performing. This increased pacing rate is sometimes referred to as the “sensor indicated rate”. When exertion has stopped the pacemaker will progressively decrease the paced rate down to the lower rate.

Through careful use of small refinement steps and appropriate intermediate abstractions, we were able to achieve an impressive degree of automatic proof. Here in this section we have also mentioned the table of proof obligations for single electrode pacemaker.

All the proof obligations for all five levels were generated and proved using the RODIN proof tool. The statistics from the mechanical proof effort for all of the refinement levels are outlined in Table-2. In the table, the total number of POs column represents the total number of proof obligations generated for each level. The Interactive Proof column represents the number of those proof obligations that had to be proved interactively. Those proof obligations that were not proved interactively were proved completely automatically by the prover.

The complete development of single electrode pacemaker resulted in 338(100%) proof obligations, in which 300(88%) were proved completely automatically by RODIN tool. The remaining 38(12%) proof obligations were proved interactively using RODIN tool. This refinement approach together with the RODIN tool supports an incremental style of system development. We have presented the complete refinements in top down manner. We started with the highest level specification and then produced a model approximating the lowest level. However in attempting to prove refinement between these models it was clear that the abstraction gap was too large would have required a complex gluing invariant. Instead we decided that some intermediate abstraction was required. Any modifications to the refinement model had an impact on the existing proofs. For that we have need to give the proper gluing invariants.

As we can see from the Table-2 in refinement first we have proved 16 proofs obligations interactively out of total no. of 62 proofs obligations. The most difficult proof was in first and third refinement of the abstract

Table 4.2: Proof obligations of all modes of single electrode pacemaker system

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	118	112(95%)	4(5%)
First Refinement	60	44(73%)	16(27%)
Second Refinement	44	40(91%)	4(9%)
Third Refinement	36	24(66%)	12(34%)
Fourth Refinement	78	78(100%)	0(0%)
Total	336	298(89%)	38(11%)

model. Most of the interactive proof generated due to adding the new behaviour threshold and hysteresis concepts into the single elctrode pacemaker system under hard real-time constraint. In the pacemaker system strong action-reaction patterns apply with real-time constraints and we have proved successfully the generated proof obligations in different situations of threshold, hysteresis and rate adapting pacing and sensing in different operational modes.

4.4 Abstract model of Pacemaker

In the abstract model of single electrode pacemaker, we introduced the basic notions of action-reaction and real-time constraint patterns. We already explained the different types of action-reaction patterns. Here, in modelling of single electrode pacemaker we applied the strong action-reaction patterns in step wise refinements in all modes. In this abstraction, we begin with an abstract model of a single electrode pacemaker system focusing on pacing and sensing modes properties and operations control the pumping rate of natural pacemaker or human heart. However, some pacing modes (AOOR,VOOR,AAIR,VVIR,AATR and VVTR) are not distinguished in this level. Instead they are emerged to next refinement given in Section 10. Thus, in this level, for every modes of pacemaker are treated in the same way as common basic modes, which are essential for the single electrode pacemaker.
The model consists of several modules, each corresponding to an operating mode of the pacemaker but some operating modes get through the refinements of model.

4.4.1 Abstraction of AOO and VOO modes

For modeling the AOO and VOO modes of single electrode pacemaker use the constant parameters as axioms as follows:

$$axm1 : LRL \in 30 \,..\, 175 \land LRL = 60$$
$$axm2 : URL \in 50 \,..\, 175 \land URL = 120$$
$$axm3 : URL_Time_Int \in \mathbb{N}_1 \land URL_Time_Int = 60000/URL$$
$$axm4 : LRL_Time_Int \in \mathbb{N}_1 \land LRL_Time_Int = 60000/LRL$$

We introduced the constants LRL and URL that relate to the Lower Rate Limit (minimum number of pace pulses delivered per minute) and Upper Rate Limit (How fast the pacemaker will allow the heart to be paced). The numeric conversions are needed because the time unit is milliseconds. The two new constants (URL_Time_Int) and (LRL_Time_Int) represent the corresponding URI (upper rate interval) and LRI (lower rate interval) respectively. The interval unit is the "millisecond" (ms).
The variable ($Pacemaker_Actuator$) enumerated types representing the presence or absence of a pulse at single electrode pacemaker where electrode can be either do nothing or discharge a pulse on the atria or ventricular. The next variable (sp) (since pace) is a natural number that represents counting timeline between two consecutive pace and it is also the counter of clock which value is always less then and equal to LRI. The variable ($last_sp$) (last value of since pace or clock counter) represents the last counting timeline between consecutive pace. The ($Pace_Int_set$) introduces as a flag variable to change the value of Pace interval. The ($Pace_Int$) variable is modify by the programmer in beginning and set the interval value for two consecutive pace.

$inv1 : Pacemaker_Actuator \in status$
$inv2 : sp \in \mathbb{N}$
$inv3 : sp \leq LRL_Time_Int$
$inv4 : last_sp \in \mathbb{N}$
$inv5 : Pace_Int_set \in BOOL$
$inv6 : Pace_Int \in URL_Time_Int \,..\, LRL_Time_Int$
$inv7 : last_sp \geq URL_Time_Int \land last_sp \leq LRL_Time_Int$
$inv8 : Pace_Int_set = FALSE \land sp > 0 \land sp < Pace_Int$
$\Rightarrow$
$Pacemaker_Actuator = OFF$
$inv9 : Pace_Int_set = FALSE \land sp > Pace_Int$
$\Rightarrow$
$Pacemaker_Actuator = ON$

The invariant ($inv7$) represents that the value of pace interval ($Pace_int$) should be between upper rate interval to lower rate interval. The next invariants ($inv8$ and $inv9$) represent the safety properties of single electrode pacemaker and indicate that the pacemaker should pace only and only after pace interval. Pacemaker actuator should be never activated in between the pace interval.
In the single electrode pacemaker the pacemaker either paced in atria or ventricular in the modes of AOO and VOO respectively. The above described all axiom and constants are common for AOO and VOO modes. We have introduced the new events and variables in forthcoming models as refinement in the incremental development of the the single electrode pacemaker system. In abstract specification of the pacemaker modes include events modeling, pacing into the heart within time constraints, stop the pacing into the heart within time constraint and progressive increments in the clock cycle to control all the atomic events of pacemaker. There are four significant events in our abstract model of AOO and VOO modes as follows:-
The event ($Pace_ON$) represents the pacing operation of the single electrode pacemaker into the heart either in atrial chamber using AOO mode or in ventricular chamber using VOO pacing mode. The guard ($grd1$) states that the pacemaker actuator should be in OFF state and next guard ($grd2$) states that clock counter (sp) should satisfy the condition $sp \geq Pace_Int$. When guard of event satisfy then action will take the effect and pacemaker will discharge the pulse to the heart and assign the value of clock counter variable (sp) to another last clock counting variable ($last_sp$).

EVENT Pace_ON
WHERE
grd1 : $Pacemaker_Actuator = OFF$
grd2 : $sp \geq Pace_Int$
THEN
act1 : $Pacemaker_Actuator := ON$
act2 : $last_sp := sp$
END

The event $Pace_OFF$ used to stop the pulse discharging to the heart and set the value "1" to current clock counter variable (sp). The guard ($grd1$ and $grd2$) of this event state that the pacemaker should be in "ON" state and clock counter value should be grater then ($Pace_Int$) variable.

EVENT Pace_OFF
WHERE
grd1 : $Pacemaker_Actuator = ON$
grd2 : $sp + 1 > Pace_Int$
THEN
act1 : $Pacemaker_Actuator := OFF$
act2 : $sp := 1$
END

The event (tic) is an important event which control the all other events of pacemaker. The guard $(grd1)$ states that the value of clock counter should be in between 1 to $Pace_Int$. The action of this event progressively increase the value of clock counter within time limit constraint.

EVENT tic
  WHERE
    grd1 : $sp > 0 \wedge sp + 1 \leq Pace_Int$
  THEN
    act1 : $sp := sp + 1$
  END

The event Set_Pace_Int used as keep event in abstract model for choosing the value of $(Pace_Int)$ variable. The value of variable $Pace_Int$ can be only changed when the flag variable $Pace_Int_set$ will be TRUE.

EVENT Set_Pace_Int
  WHERE
    grd1 : $Pace_Int_set = TRUE$
  THEN
    act1 : $Pace_Int$
      $: |$
      $(Pace_Int' \in URL_Time_Int \,..\, LRL_Time_Int)$
  END

4.4.2 Abstraction of AAI and VVI modes

In the abstract model of AAI mode all the constants and variables are common as AOO mode. We introduced a new constant ARP(Atrial Refractory Period) which represents a period during which pacemaker timing in atrial will not be affected by events that occur with in(no sensing with initiation of a new Lower Rate Interval). Similarly in the abstract model of VVI we introduced the new constants VRP (Ventricular Refractory Period) which represents a period during which pacemaker timing in ventricular will not be affected by events that occur with in(no sensing with initiation of a new Lower Rate Interval). We added two new variables $(Pacemaker_Sensor)$ and $(last_ss)$ in the abstract model of AAI and VVI mode. The variable $(Pacemaker_Sensor)$ enumerated types representing the presence and absence of a pulse in single electrode pacemaker, where electrode can be sense nothing or sense the pulse signal from the atria or ventricular. The following constants and invariants are adding in AAI and VVI modes abstraction:-

$axm1 : ARP \in 150 \,..\, 500 \wedge ARP = 250$
$axm2 : VRP \in 150 \,..\, 500 \wedge VRP = 320$
$inv1 : Pacemaker_Sensor \in status$
$inv2 : last_ss \in \mathbb{N}$
$inv3 : last_ss \geq ARP \wedge last_ss \leq LRL_Time_Int$
$inv4 : last_ss \geq VRP \wedge last_ss \leq LRL_Time_Int$

In this abstract model the event $(Pace_ON)$ is similar to $(Pace_ON)$ event of AOO and VOO modes. One new event $(Pace_OFF_with_Sensor)$ added in the abstraction of AAI or VVI modes and some new guards and actions added in all other events of AOO and VOO modes. In the event $(Pace_OFF)$ of AAI and VVI modes we added the new guard $(grd2)$ which states that pacemaker sensor should be in ON state and action part states that the sensor will stop the sensing of the pulse from atria or ventricular.

EVENT Pace_OFF
  WHERE
    $\oplus$ $grd3 : Pacemaker_Sensor = ON$
  THEN
    $\oplus$ $act2 : Pacemaker_Sensor := OFF$
  END

The event $(Pace_OFF_with_Sensor)$ is a new event in this abstraction of AAI and VVI modes. The guards $(grd1, grd2$ and $grd3)$ state that when pacemaker actuator is OFF, pacemaker sensor is ON and counter of clock is greater than ARP or VRP then the actions state that it store the value of clock counter (sp), reset the clock counter and stop the pacemaker sensor for sensing of atria or ventricular. The LRI consists of two portions, the ventricular refractory period (VRP), and the alert period. The VRP is initiated at the start of the LRI with each sensed or paced event. It is a period during which pacemaker timing will not be affected by events that occur within it. The alert period follows and is the interval during which sensing can occur, inhibit pacing, and initiate a new LRI.

EVENT Pace_OFF_with_Sensor
WHERE
grd1 : $Pacemaker_Actuator = OFF$
grd2 : $Pacemaker_Sensor = ON$
grd3 : $sp \geq ARP$
THEN
act1 : $last_ss := sp$
act2 : $sp := 1$
act3 : $Pacemaker_Sensor := OFF$
END

In the abstract model of AAI and VVI modes only modify the guard of tic event. The action part of this event is remain same as previous abstraction of AOO and VOO modes. The modified guard has been given as follows:-

EVENT tic
WHERE
grd1 : $(sp > 0 \wedge sp + 1 \leq ARP)$
$\vee$
$(sp + 1 > ARP \wedge sp + 1 \leq Pace_Int \wedge$
$Pacemaker_Sensor = ON)$
THEN
END

The event $(Sense_ON)$ is a new event of pacemaker AAI and VVI modes. This event used to start the sensing process of pacemaker's sensor on the basis time constraints. The guards of this event state that pacemaker should be in OFF state and progessive clock counter (sp) should be greater than ARP inetrval and less than pace interval. The action of this event states that pacemaker sensor will be in ON state when all guards will satisfy.

EVENT Sense_ON
WHERE
grd1 : $Pacemaker_Sensor = OFF$
grd2 : $sp \geq ARP$
grd3 $sp < Pace_Int$
THEN
act1 : $Pacemaker_Sensor := ON$
END

4.4.3 Abstraction of AAT and VVT modes

In the abstract model of AAT and VVT modes, all the constants and variables are similar to AAI and VVI modes respectively. Similarly all the events of AAT and VVT modes same as AAI and VVI modes but a new event $(Pace_ON_with_Sensor)$ used in place of $(Pace_OFF_with_Sensor)$ event. The guards $(grd1, grd2$ and $grd3)$ state that when pacemaker actuator is OFF, pacemaker sensor is ON and counter of

clock is grater than ARP or VRP then the actions state that it store the value of clock counter (sp) and start the pacemaker sensor for sensing of atria or ventricular. This event triggers pacing in atria or ventricular when sense the pulse from atria or ventricular chamber in alert period (LRI-VRP or LRI-ARP). The alert period follows and is the interval during which sensing can occur, triggers pacing, and initiate a new LRI.

EVENT Pace_OFF_with_Sensor
WHERE
grd1 : $Pacemaker_Actuator = OFF$
grd2 : $Pacemaker_Sensor = ON$
grd3 : $sp \geq ARP$ **THEN**
act1 : $Pacemaker_Sensor := ON$
act2 : $last_ss := sp$
END

4.5 First refinement

In the abstract model we have presented that single electrode pacemaker pacing and sensing in atomic step in natural pacemaker or heart under real time constraints. So our goal is to model pacing and sensing of pacemaker in correct manner. In the first refinement step, we introduce the more and more invariants in different operating modes of pacemaker to apply the strong action-reaction and real-time patterns. In AOO and VOO modes we have already add the strong invariants in abstarct model so we have not need to add any extra invariants but we are adding new invariants in AAI and AAT modes as follows.

$inv1 : sp > 0 \wedge sp < ARP \Rightarrow Pacemaker_Sensor = OFF$
$inv2 : sp > ARP \wedge sp \leq Pace_Int \Rightarrow Pacemaker_Sensor = ON$
$inv3 : sp > 0 \wedge sp < ARP \Rightarrow Pacemaker_Actuator = OFF$

The modes of VVI and VVT is same as AAI and AAT modes respectively, there is difference between only in atria refractory period and ventricular refractory period. In the VVI and VVT modes we have used the same invariants which defined above, but we used the VRP in place of ARP in the invariants of VVI and VVT modes. The invariant $(inv1)$ states that the pacemaker sensor will never sense the value of pacemaker pulse during the ARP or VRP time period. The invariant $(inv2)$ states that the pacemaker sensor will be in ON state or continuously sensing the value from heart chamber within an alert period (LRI-ARP or LRI-VRP). The last invariant $(inv3)$ states that pacemaker actuator also should be stop within the ARP or VRP period. Hence there is no pacing and sensing activities by single electrode pacemaker within the atria or ventricular refractory period. We have done the following changes in the guard of tic event and it controls the progressive increment in the clock counter.

$$grd1 : (sp > 0 \wedge sp+1 \leq ARP \wedge sp+1 \leq Pace_Int \wedge Pacemaker_Sensor = OFF \wedge Pacemaker_Actuator = OFF) \vee (sp \geq ARP \wedge sp+1 \leq Pace_Int \wedge Pacemaker_Sensor = ON \wedge Pacemaker_Actuator = OFF)$$

4.6 Second refinement:Threshold

The basic requirements of the single electrode pacemaker system is pacing and sensing into the natural pacemaker or heart in any particular chamber in atria or in ventricular. In the stepwise refinement of abstract model we introduced the concept of sensing threshold value of the single electrode pacemaker. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of a heart and a pulse generator for deliverying stimulation pulses to the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value and a safety margin. The constant THR hold the constant value of atria chamber as follows:-

$$axm1 : THR \in \mathbb{N}1 \wedge THR = 75$$

The constant THR hold the value of ventricular chamber as follows:-

$$axm1 : THR \in \mathbb{N}1 \wedge THR = 250$$

In the pacemaker sensor start to sensing after a particular time interval but the pacemaker actuator will take effect when sensing threshold value of sensor will be greater then the standard threshold value. In this refinement each time the pacemaker sensor sense the pulse signal either from atria or ventricular.
The following invariants are given for different modes of pacemaker. The invariant $(inv1)$ states that pacemaker actuator will be in OFF state when pacemaker sensor will sense the pulse value from the atria chamber, the sensed value is larger then sensing threshold value, the value of since pace time counter (sp) is greater than atria refractory period (ARP) and less than pacing interval and state of threshold value is TRUE. The invariant $(inv2)$ states that pacemaker actuator will be in OFF state when pacemaker sensor will sense the pulse value from the ventricular chamber, the sensed value is larger then sensing threshold value, the value of since pace time counter (sp) is greater than ventricular refractory period (VRP) and less than pacing interval and state of threshold value is TRUE. The third invariant $(inv3)$ states that pacemaker actuator will be in ON state when pacemaker sensor will sense the pulse value from the atria chamber, the sensed value is larger then sensing threshold value, the value of since pace time counter (sp) is greater than atria refractory period (ARP) and less than pacing interval and state of threshold value is TRUE. Similarly the last invariant $(inv4)$ states that pacemaker actuator will be in ON state when pacemaker sensor will sense the pulse value from the ventricular chamber, the sensed value is larger then sensing threshold value, the value of since pace time counter (sp) is greater than ventricular refractory period (VRP) and less than pacing interval and state of threshold value is TRUE. The threshold value of different chambers(atria and ventricular) in different modes(AAI,VVI,AAT and VVT) are specified by the doctor after diagnose the patient requirements.

$$\begin{array}{ll}
inv1 : & sp > ARP \wedge Pacemaker_Sensor = ON \wedge thr \geq THR \wedge \\
& sp < Pace_Int \wedge thr_val_state = TRUE \\
& \Rightarrow \\
& Pacemaker_Actuator = OFF \\
inv2 : & sp > VRP \wedge Pacemaker_Sensor = ON \wedge thr \geq THR \wedge \\
& sp < Pace_Int \wedge thr_val_state = TRUE \\
& \Rightarrow \\
& Pacemaker_Actuator = OFF \\
inv3 : & sp > ARP \wedge Pacemaker_Sensor = ON \wedge thr \geq THR \wedge \\
& sp < Pace_Int \wedge thr_val_state = TRUE \\
& \Rightarrow \\
& Pacemaker_Actuator = ON \\
inv4 : & sp > VRP \wedge Pacemaker_Sensor = ON \wedge thr \geq THR \wedge \\
& sp < Pace_Int \wedge thr_val_state = TRUE \\
& \Rightarrow \\
& Pacemaker_Actuator = ON
\end{array}$$

The new variable (thr) introduced in this refinement and we have added this variable in different events of last refinement. We have added the guard $(grd4 : thr \geq THR$ in events $(Pace_OFF_with_Sensor, Pace_ON_with_Sensor)$ and we have modified the guard of (tic) event as follows:-

$$\begin{array}{ll}
grd1 : & (sp > 0 \wedge sp + 1 \leq ARP \wedge sp + 1 \leq Pace_Int \wedge \\
& Pacemaker_Sensor = OFF \wedge Pacemaker_Actuator = OFF) \\
& \vee \\
& (sp \geq ARP \wedge sp + 1 \leq Pace_Int \wedge Pacemaker_Sensor = ON \wedge \\
& Pacemaker_Actuator = OFF \wedge thr < THR \wedge thr_val_state = FALSE)
\end{array}$$

In the ventricular chamber we used the VRP constant in place of ARP constant in gaurd of (tic) event and modified guard is same for all other events of different modes(AAI,VVI,AAT and VVT) of pacemaker. In the refinement of event $(Sense_ON)$, we have added the new action as $(thr_val_state := TRUE)$. This action used to change the state of threshold variable as TRUE. The new event (Thr_value) introduced in all modes (AAI,VVI,VVI and VVT), which used to read the value of pacemaker sensor. The guards of this event state that the pacmaker sensor should be in ON state and since pace time counter (sp) should be greater than atria refractory period(ARP) or ventricular refractory period(VRP) and less than pace interval $(Pace_Int)$ and state of threshold value (thr_val_state) should be in TRUE state. When all guards of this event satisfy then the sensed value will assign to the threshold variable (thr) and set the threshold value state (thr_val_state) as FALSE.

EVENT Thr_value
ANY
th
WHERE
grd1 : $Pacemaker_Sensor = ON$
grd2 : $th \in \mathbb{N}$
grd3 : $sp \geq ARP \vee sp \geq VRP$
grd4 : $thr_val_state = TRUE$
grd5 : $sp < Pace_Int$
THEN
act1 : $thr := th$
act2 : $thr_val_state := FALSE$
END

4.7 Third refinement:Hysteresis

In the third refinement, we have introduced the concept of "$Hysteresis''$ in pacing and sensing mode of single electrode pacemaker. "Hysteresis", from the Greek for "to lag behind," means a delay of effect behind the cause. In pacemakers, this means delaying pacing to maximize patient benefit. The application of a hysteresis interval to provide consistent pacing of the atrial or ventricle, or to prevent constant pacing of the atrial or ventricle. An implantable pacemaker system is provided with a conditional hysteresis feature, whereby a hysteresis value is added to the pacing escape interval (Hyt_Pace_Int) only when the prior spontaneous rate corresponded to a rate below the top of a predetermined hysteresis band. This feature limits the lengthening of the escape interval (Hyt_Pace_Int) when there are sudden drops in the natural rate thereby avoiding excessive changes in rate. In a preferred embodiment, the pacemaker defines a hysteresis band around a given pacing rate, lower rate limit, the band having an upper hysteresis limit (URL_Time_Int) and a lower hysteresis limit (HRL_Time_Int). No hysteresis lengthening of the escape interval is utilized for spontaneous heartbeats having rates above the upper hysteresis limit; for spontaneous heartbeats having rates between the lower rate limit and the upper hysteresis limit, an escape interval is set to have a value corresponding to a rate between the pacing limit and the lower rate limit of the hysteresis band which is below the lower rate limit; and for a sensed spontaneous rate below the lower rate limit, a hysteresis escape interval corresponding to the lower hysteresis limit is established. In the preferred embodiment, sensed heartbeats having a prior rate between the lower rate limit and the upper hysteresis limit cause an escape interval which is lengthened beyond the LRL escape interval by an amount which varies linearly with the differential between the upper hysteresis rate limit and the spontaneous rate. We introduced the new constants for modeling the $Hysteresis$ concepts in the modes of pacemaker as follows:-

$axm1 : HRL = LRL$
$axm2 : HRL_Time_Int = LRL_Time_Int$
$axm3 : Hyt_Pace_Int = HRL_Time_Int$
$axm4 : HYT_State \in BOOL$

In the above axioms, the items $(axm2, axm3)$ introduced the constants for describing the hysteresis interval limits and the last item $(axm4)$ introduced the hysteresis state in this refinement. We introduced the three

invariants and one theorem in this refinement as follows:-

$$
\begin{aligned}
&inv1 : HYT_State = TRUE\\
&\qquad \Rightarrow\\
&\qquad last_sp \geq URL_Time_Int \wedge last_sp \leq HRL_Time_Int\\
&inv2 : HYT_State = TRUE\\
&\qquad \Rightarrow\\
&\qquad last_ss \geq ARP \wedge last_ss \leq HRL_Time_Int\\
&inv3 : HYT_State = TRUE\\
&\qquad \Rightarrow\\
&\qquad Pace_Int = HRL_Time_Int\\
&thm1 : HYT_State = FALSE\\
&\qquad \Rightarrow\\
&\qquad Pace_Int \geq URL_Time_Int \wedge Pace_Int \leq HRL_Time_Int
\end{aligned}
$$

The invariant ($inv1$) states that if hysteresis state is TRUE then interval between two pace should be in hysteresis band (upper rate limit to lower rate limit). The next invariant ($inv2$) states that if hysteresis state is TRUE then the interval between two sensed pulse should be greater than ARP and less then lower hysteresis rate limit (HRL_Time_Int). The third invariant ($inv3$) states that if hysteresis state is TRUE then pacing interval ($Pace_Int$) and lower hysteresis rate limit (HRL_Time_Int) should be equal. The theorem ($thm1$) states that if hysteresis state is FALSE then pacing inetrval should be greater than upper rate limit time interval (URL_Time_Int) and less than hysteresis rate limit time interval (HRL_Time_Int). In this refinement the invariants and theorem is same for all the modes(AAI and VVI) but in VVI modes we applied the VRP in place of ARP in invariant ($inv2$). Many VVI and AAI modes of pacemakers have a rate function called $hysteresis$. $Positive$ hysteresis can add an additional period of time for the pacemaker to wait and see if a native R wave will occur before pacing. In this application it can occur only after an R wave is sensed and does not occur after a paced event. The hysteresis rate is less than the lower rate. In this manner the principal purpose of hysteresis is to allow the patient to have his or her own underlying rhythm as much as possible. This can help conserve the pacemaker's battery life. $Hysteresis$ concept is not available in AAT and VVT modes of the pacemaker. But in the refinement we modeled the hysteresis concept for AAT and VVT modes and it satisfy all the proof obligations which occured in this refinement. We have checked it that there in no any effect in AAT and VVT modes of pacemaker when applied the $hysteresis$. So $hysteresis$ is only aplicable with AAI and VVI modes.

We have't introduced any extra events in this refinement. We have added the hysteresis related constants and variables in already defined events. We have added the following new guard ($grd4$) in event ($Pace_ON$) and ($grd5$) in events ($Sense_ON$) and (Thr_value). These guards represent that hysteresis states (ON and OFF),hysteresis pacing inetrval and normal pace interval of the pacemaker parameters should be valid at different operating modes of the pacemaker in pacemaker events.

$$
\begin{aligned}
&grd4 : (HYT_State = FALSE \wedge sp \geq Pace_Int)\\
&\qquad \vee\\
&\qquad (HYT_State = TRUE \wedge sp \geq Hyt_Pace_Int)\\
\\
&grd5 : (HYT_State = FALSE \wedge sp < Pace_Int)\\
&\qquad \vee\\
&\qquad (HYT_State = TRUE \wedge sp < Hyt_Pace_Int)
\end{aligned}
$$

We have also modified the old guard of event (tic) with the following new guard. This guard is necessary for satisfy the time constraints for every operation of the pacemaker. The modified guard controls the time counter in different operating modes of pacemaker. This modified guard is similar for AAI and VVI modes but in VVI modes we have used the ventricular refractory period (VRP) in place of atria refractory period (ARP).

$$
\begin{array}{l}
grd1: ((HYT_State = FALSE \wedge sp > 0 \wedge sp + 1 \leq ARP \wedge \\
\quad sp + 1 \leq Pace_Int \wedge Pacemaker_Sensor = OFF \wedge \\
\quad Pacemaker_Actuator = OFF) \\
\quad \vee \\
\quad (HYT_State = FALSE \wedge sp \geq ARP \wedge sp + 1 \leq Pace_Int \wedge \\
\quad Pacemaker_Sensor = ON \wedge Pacemaker_Actuator = OFF \wedge \\
\quad thr < THR \wedge thr_val_state = FALSE)) \\
\quad \vee \\
\quad ((HYT_State = TRUE \wedge sp > 0 \wedge sp + 1 \leq ARP \wedge \\
\quad sp + 1 \leq Hyt_Pace_Int \wedge Pacemaker_Sensor = OFF \wedge \\
\quad Pacemaker_Actuator = OFF) \\
\quad \vee \\
\quad (HYT_State = TRUE \wedge sp \geq ARP \wedge sp + 1 \leq Hyt_Pace_Int \wedge \\
\quad Pacemaker_Sensor = ON \wedge Pacemaker_Actuator = OFF \wedge \\
\quad thr < THR \wedge thr_val_state = FALSE))
\end{array}
$$

4.8 Fourth refinement:Rate Modulation

This refinement is the last and important refinement in the single electrode pacemaker system. In this refinements we introduced the rate responsive technique to the pacemaker. Rate responsive term has led to the more acceptable use of the terms rate adaptive and rate modulating. All these terms are used to describe the capacity of a pacing system to respond to physiologic need by increasing and decreasing pacing rate. The capability of a pacing system depends on the presence of one of a variety of physiologic sensors that monitor need or indication for rate variability. The predominant need for rate modulation derives from physical activity or exertion. There are other physiologic situations in which normally there are modulations of heart rate for example, with fever and emotional stress. These, however, are substantially less important, especially in the context of pacing systems. This refinement of pacemaker also give some new pacing and sensing mode (AAIR,VVIR,AATR and VVTR) of the pacemaker. The rate adapting mode of pacemaker can progressively pace faster than the lower rate, but no more than the upper sensor rate limit, when it determines that heart rate needs to increase. This typically occurs with exercise in patients that cannot increase their own heart rate. The amount of rate increase is determined by how much exertion the pacemaker thinks the patient is performing. This increased pacing rate is sometimes referred to as the "sensor indicated rate". When exertion has stopped the pacemaker will progressively decrease the paced rate down to the lower rate. For modeling the rate modulation technique in single electrode pacemaker, we introduced the some axioms as follows:-

$$
\begin{array}{l}
axm1: MSR \in 50 \,..\, 175 \wedge MSR = 120 \\
axm2: threshold \in \mathbb{N}_1 \wedge threshold = 4 \\
axm3: reactionTime \in 10 \,..\, 50 \wedge reactionTime = 10 \\
axm4: recoveryTime \in 2 \,..\, 16 \wedge recoveryTime = 2 \\
axm5: responseFactor \in 1 \,..\, 16 \wedge responseFactor = 8
\end{array}
$$

In above axioms, $(axm1)$ represents the maximum sensor rate (MSR) is maximum pacing rate allowed as a result of sensor control and it should be in between 50 to 175 ppm(pulse per minute). We have taken the nominal value of MSR in this model as 120 ppm. The next axiom $(axm2)$ represents the activity threshold is the value the accelerometer sensor output shall exceed before the pacemaker's rate is affected by activity data. The nominal value of activity threshold is 4 in this model. The accelerometer shall determine the rate of increase of the pacing rate. The reaction time is the time required for an activity to drive the rate from LRL to MSR, which is defined as axiom $(axm3)$. Similarly axioms $(axm4, axm5)$ represent the recovery time and response factor in rate adapting pacing respectively. The recovery time shall be the time required for the rate to fall from MSR to LRL when activity falls below the activity threshold and the response factor in rate adapting pacing, the accelerometer shall determine the pacing rate that occurs at various levels of steady state patient activity. The highest response factor setting (16) shall allow the greatest incremental change in rate and the lowest response factor setting (1) shall allow a smaller change in rate. We have taken the nominal value of reaction time, recovery time and response factor in our model.

We introduced the new variable ($acler_sensed$) as $acler_sensed \in \mathbb{N}$, to store the sensed value from the chamber using the pacemaker sensor. We have introduced the following invariants as follows in this refinement:-

$inv1 : acler_sensed < threshold \Rightarrow Pace_Int = 60000/LRL$
$inv2 : acler_sensed > threshold \Rightarrow Pace_Int = 60000/MSR$

The invariant ($inv1$) states that when the sensed value of the accelerometer sensor is less than constant activity threshold value then the pacing inetrval should be equal to $60000/LRL$ so that the heart rate never fall below the lower rate limit (LRL) and similarly the invariant ($inv2$) states that when sensed value of the accelerometer sensor is greater than constant activity threshold value then the pacing interval should be equal to $60000/MSR$, so that the heart rate never exceed the maximum sensor rate or upper rate limit of the heart pacing. These two invariants always check the safety margin in rate adapting pacing. Finally the simulation of the rate controller follows as a relation between the reach of the MSR with a exceeding input value of the treshold, and the LRL as a decrease after the reacovery time form the MSR or the normal functioning of the system.
In this final refinement we introduced only two extra events ($Increase_Interval$, $Decrease_Interval$) to control the pacing rate of the single electrode pacemaker in rate adapting pacing modes. The new event ($Increase_Interval$) controls the value of pace interval whenever the sensed value of accelerometer sensor is greater then activity threshold value. The other new event ($Decrease_Interval$) controls the value of pace interval whenever the sensed value of accelerometer sensor less then activity threshold value. After introduced the these two new events in this refinement, we have found the new modes(AOOR, VOOR, AAIR, VVIR, AATR and VVTR) in the single electrode pacemaker, which are using in the pacemaker as a rate adaptive pacing features. All these modes apply to control the pacing activity of pacemaker. So here we have found the relationship between different modes of pacemaker in stepwise refinements. Two new events introduced in this refinement are essential for controlling the rate adapting pacing modes of the single electrode pacemeker.

EVENT Increase_Interval
ANY
WHERE
grd1 : $acler_sensed > threshold$
THEN
act1 : $Pace_Int := 60000/MSR$
END

EVENT Decrease_Interval
ANY
WHERE
grd1 : $acler_sensed < threshold$
THEN
act1 : $Pace_Int := 60000/LRL$
END

We introduced the similar refinements in all other modes of the single electrode pacemaker for atria as well as ventricular chambers. Rate modulated pacemakers mimic physiological response by increasing heart rate and, subsequently, cardiac output in response to exercise. Rate modulated pacemaker use metabolic or motion-derived sensors to adjust pacing rate based on physiologic requirements by translating indices of increased metabolic need into signals that can be used to restore chronotropic competence. The most common and versatile sensors include activity or acceleration, minute ventilation, or combinations; these sensors remain functional with standard pacing leads. All sensor systems have some limitations. Although, any rate response is better than none,the degree of rate response depends on programming. For a single activity level, any sensor can be programmed to provide virtually any desired rate. Different sensors respond differently to the same stimuli. Hence, combinations of complementary sensors may better simulate normal

sinus node response. Some devices automatically reprogram rate response parameters based on average activity levels. Finaly we have modeled the all modes of single electrode pacemaker with all required features of different modes of pacemaker system using stepwise refinements.

4.9 Model Validation using ProB

A systematic testing approach was used to validate the models derived during the staged development process. "Validation" in this context refers to the activity of gaining confidence that the formal models developed are consistent with the requirements expressed in the requirements document [68]. We have used the ProB [89] validation tool to test the all scenarios of all modes of single electrode pacemaker. The pacemaker specification was developed and formally proven by the Event-B. However, the development contains certain assumptions about the actual single electrode pacemaker system which have to be validated separately in order to ensure safe operation. We have used the ProB to dig all required information and missing safety properties in the model. We have used the ProB to test all modes scenarios of the single electrode pacemaker in different interesting situations such as the absence of input pulses, hysteresis, threshold and rate adapting pacing. The validation process involves to sense the chamber and paced into the chamber at the correct time in different situations such as hysteresis and rate adapting modes. We have successfully tested the all cases of pacemaker modes after modeling in Event-B, using ProB validation tool.

4.10 Conclusion

The stepwise development of single electrode pacemaker system help us to discover the exact behaviour of pacing and sensing activities. Our objective in the work reported here was to assess the feasibility of using an incremental approach in the production of a useful model of a realistic real-time action-reaction system as a pacemaker. The pacemaker case study suggests that such an approach can yield a viable model that can be subjected to useful validation against system-level properties at an early stage in the development process. We have applied the action-reaction pattern [2] and time based pattern [70] to developed the pacemaker system modes. In our study of pacemaker system we have modeled the 12 modes of single electrode pacemaker with different situations such as hysteresis and rate adaptive pacing. We have also discover the relationship between different modes of the pacemaker in stepwise refinements Fig . The refinements gradually introduce the various invariants of the system. The proof leads us to the discovery of the confirmation event to get the complete correctness, which was not the case of the I/O automata modelling. Our approach has been very pragmatic, driven by the aim of providing a fully formal modelling approach with a low barrier to industrial adoption. We have not yet dealt with the relationship between the incremental addition of detail and formal refinement. We have outlined how an incremental refinement approach to the single elctrode pacemaker system allowed us to achieve a very high degree of automatic proof. The powerful support provided by the rodin tool was essential to achieving what we believe was a very successful development. Rodin proof was used to generate the hundreds of proof obligations and to discharge those obligations automatically and interactively. Without this level of automated support, making the changes to the refinement chain that we did make would have been far too tedious. Our approach is the methodology of separation of concerns: first prove the basic behaviour of singlr electrode pacemaker system at an abstract level; then, and only then, gradually introduce the peculiarity of the specific properties. What is important about our approach is that the fundamental properties we have proved at the beginning, namely the reachability and the uniqueness of a solution, are kept through the refinement process (provided, of course, the required proofs are done). We have validated the single electrode pacemaker system using the ProB validation tool and find the correctness of our proved single electrode pacemaker system within the real-time constraint.

Chapter 5

Formal Development of Two-Electrode Cardiac Pacing System

Sommaire

5.1 Introduction . . . 63
5.2 Basic Overview of Pacemaker system . . . 64
5.2.1 The Heart System . . . 64
5.2.2 The Pacemaker system . . . 65
5.3 The modelling framework . . . 66
5.3.1 Modelling actions over states . . . 66
5.3.2 Model refinement . . . 67
5.3.3 Guidelines for EVENT B Modelling . . . 69
5.4 Formal Development . . . 70
5.4.1 The Context and Initial Model . . . 70
5.4.2 First refinement:Threshold . . . 94
5.4.3 Second refinement:Rate Modulation . . . 105
5.5 Model Validation and Analysis . . . 106
5.6 Conclusion and Future Works . . . 107

To build a high quality and zero defects medical devices and softwares is a crucial task. Formal modeling techniques help to achieve this target at certain level. Formal modeling of High-Confidence Medical devices those are too much error prone in operating, are an International Grand Challenge in the area of Verified Software. Formal modeling of an artificial pacemaker is also one of the proposed challenge. The architecture and functional behaviour of the double electrode pacemaker is more complex than the single electrode pacemaker. Proof-based an incremental approach, we use to develop the formal model of functional behaviour of the double electrode pacemaker. The incremental proof-based development is mainly driven by the refinement between an abstract model of the system and its detailed design through a series of refinements, which adds parametric based functional properties to the abstract system-level specifications using some intermediate models. The properties express system architecture and action-reaction under real-time constraints. This technical report focuses on the formal development of the double electrode operating modes and finds the common architecture of operating modes in tree form that helps to make the consistent system. The EVENT B modeling language is used to express the double electrode pacemaker and generated proof obligations are proved by RODIN platform. Finally, the pacemaker model has been validated by an EVENT B animator; ProB tool.

5.1 Introduction

The high confidence medical devices are highly dependent on the performance, the need for absolute precision can be a life or death issue. So, after a long time due to many failure cases and untrustworthiness of the medical devices, the equipment manufacturers have turned towards formalism in the engineering of medical device. For decades, software failures have costed billions of dollars a year [92]. Software failures and lack of warranties of products have emerged the software crisis. Due to software crisis, various formalism and rigourous techniques (VDM, Z, Event-B, Alloy etc.) have been used in the development process of safety-critical systems. These approaches provide the certain level of reliability and confidence to develop the error free systems. Formal methods and their tools have achieved a certain level of usability that could be applied even in industrial scale applications allowing software developers to provide more meaningful guarantees to their projects.

The high confidence medical devices are too complex in operating and several concurrent process are running together. To validate such kind of system, only simulation and testing can be usual techniques. By nature, testing can be applied only after a prototype implementation of the system has been realized. Formal verification, as opposed to testing, works on models (rather than implementation) and amounts of mathematical proofs provide correctness of a given system that can be realized the actual system in early stage of development.

Tony Hoare suggested a Grand Challenges for Computing Research [79] to integrate the research community to work together towards a common goal, agreed to be valuable and achievable by a team effort within a predicted timescale. Verification Grand Challenges is one of them. From the Verification Grand Challenges, many application areas were proposed by the Verified Software Initiative [78]. The pacemaker specification [68, 75, 57] has been proposed by the software quality research laboratory at McMaster University as a pilot project for the Verified Software Initiative [93, 85]. The challenge is characterised by system aspects including hardware requirements and safety issues. Such a system demands high integrity to achieve safety requirements.

In order to analyze the problem, we consider the triptych by D. Bjoerner [66],

where,

$\mathcal{D}, \mathcal{S} \rightarrow \mathcal{R}$
$\mathcal{D}$ = Healthcare domain
$\mathcal{S}$ = Model or chain of models of the pacemaker system
$\mathcal{R}$ = Requirements of the pacemaker system

$\mathcal{D}$ is the context of the problem to solve and it is defined in EVENT B (parameters, constants etc.). $\mathcal{S}$ is the system made up of the pacemaker and the heart. $\mathcal{R}$ is requirements for the heart system such that sensing and demand pacing under time constraints. The operating modes of Bradycardia therapy and formal model of pacemaker system are based on informal requirements, which are given by Boston Scientific [68].

H.D. Macedo, et al. [85] have developed a distributed real-time model of a cardiac pacing system but the development was not based on proof and refinement techniques. Similarly, in other case study V.P. Manna, et al. [87] have developed a simple pacemaker implementation. Recently, Gomes et al [76] wrote a formal specification of the pacemaker system using the Z modelling language. According to the paper, they have modelled the sequential model similar to H.D. Macedo et al. work [85]. In this report, we have specially covered the bradycardia operating modes of the double electrode pacemaker. We have developed the parametric and functional based incremental development of bradicardia operating modes. Moreover, we have added the *threshold*, and *rate adaptive* bradicardia operating modes. Incremental development is based on refinement approach and at every level of the development, we have proved the all required safety properties (*refinement* and *consistency checking*). Other specifications [85, 76] of the pacemaker developed as a one shot model, means those are not based on the refinement.

Our approach is based on the EVENT B modelling language which is supported by the RODIN platform integrating tools for proving models and refinements of models; moreover we use the ProB tool [89, 83] for analyzing the models and for validating these models. Here we present a stepwise development to model and verify such interdisciplinary requirements in EVENT B [69, 59]. The correctness of each step is proved in order to achieve a reliable system. The pacemaker models must be validated to ensure that they meet the requirements of the pacemaker. Hence, validation must be carried out by both formal modeling and domain

experts. The abstract model includes event modeling of bradycardia operating modes of a double electrode pacemaker system.
The refinement is supported by the RODIN [90] platform guarantees the preservation of safety properties. Thus, the behavior of the final system is preserved by an abstract model as well as in the correctly refined models. Proof-based development methods [59] integrate formal proof techniques in the development of software systems. The main idea is to start with an abstract model of the given system. Details are gradually added to this first abstract model by building a sequence of more concrete events. The relationship between two successive models in this sequence is *refinement* [59, 61]. The current work intends to explore those problems related to the modeling of bradycardia operating modes using a double electrode pacemaker system under real time constraints and to evaluate the refinement process.
The outline of the remaining report is as follows: Basic outline of a pacemaker and a heart system are given in Section 2. The modelling framework is presented in Section 3. Section 4 explores the refinement based formal development of the double electrode pacemaker. The pacemaker models are validated by the ProB tool [89, 83] and correctness of the system are analyzed by generated proof obligations (see Table-3) in Section 5. Finally, in Section 6, we conclude the report with some lessons learned from this experience and some prospective along with direction for future work.

5.2 Basic Overview of Pacemaker system

In Fig. 1 a suitable interface block diagram of the pacemaker and the heart is given. The conventional pacemakers serve two major functions, namely pacing and sensing. The pacemaker actuator is pacing by the delivery of a short, intense electrical pulse into the heart. However the pacemaker sensor uses the same electrode to detect the intrinsic activity of the heart. So, the pacemaker function of pacing and sensing activities are dependent on the behavior of the heart. The sensing and pacing functions regulates the heart rhythm. In this report, we present only the formal models of the double electrode pacemaker.

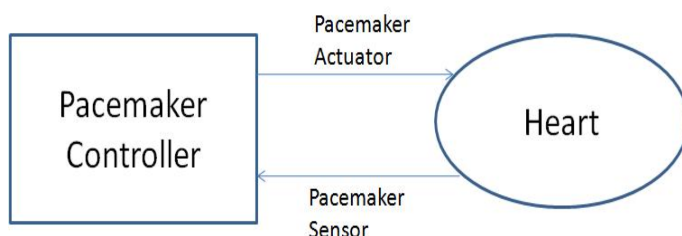


Fig. 1 Pacemaker and Heart Interface

The pacemaker system is a small electronic device that helps the heart to maintain the regular heart beat. The pacemaker is implanted in the chest during surgery. Wires called leads are put into the heart muscle. The device with the battery is placed under the skin, below the shoulder. In this study, the pacemaker is treated as an embedded system operating in an environment containing the heart. We first review the heart system that interact with the pacemaker (Section 2.1) and then consider elements of the pacemaker system itself (Section 2.2).

5.2.1 The Heart System

The human heart is wondrous in its ability to pump blood to the circulatory system continuously throughout a lifetime. The heart consists of four chambers: right atria, right ventricle, left atria and left ventricle, which contract and relax periodically. Atria form one unit and ventricles form another. The heart's mechanical system (the pump) requires at the very least impulses from the electrical system. An electrical stimulus is generated by the sinus node (see Fig. 2), which is a small mass of specialized tissue located in the right atrium of the heart. This electrical stimulus travels down through the conduction pathways and causes the heart's lower chambers to contract and pump out blood. The right and left atria are stimulated first and contract for a short period of time before the right and left ventricles. Each contraction of the ventricles represents one heartbeat.The atria contract for a fraction of a second before the ventricles, so their blood empties into the ventricles before the ventricles contract.

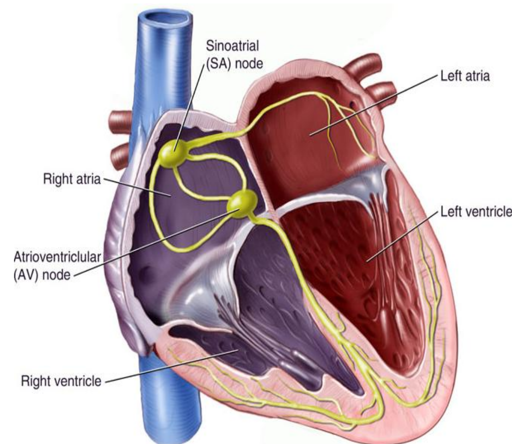


Fig. 2 Heart or Natural Pacemaker [56]

An artificial pacemaker is implanted to assist the natural pacemaker or heart in case of a arrhythmias condition to control the heart rate [86]. Arrhythmias are due to cardiac problems producing abnormal heart rhythms. In general arrhythmias reduce heamodynamic performance including situations where the heart's natural pacemaker develops an abnormal rate or rhythm or when normal conduction pathways are interrupted and a different part of the heart takes over control of the rhythm. An arrhythmia can involve an abnormal rhythm increase (tachycardia; $>$ 100 bpm) or decrease (bradycardia; $<$ 60 bpm), or may be characterized by an irregular cardiac rhythm, e.g. due to asynchrony of the cardiac chambers. The irregularity of the heartbeat, called bradycardia and techycardia. The bradycardia indicates that the heart rate falls below the expected level while in techycardia indicates that the heart rate go above the expected level of the heart rate. An artificial pacemaker can restore synchrony between the atria and ventricles. In an artificial pacemaker system, the firmware controls the hardware such that an adequate heart rate is maintained, which is necessary either because the heart's natural pacemaker is insufficiently fast or slow or there is a block in the heart's electrical conduction system [62, 73, 77, 82, 84, 86]. Beats per minute (bpm) is a basic unit to measure the rate of heart activity.

5.2.2 The Pacemaker system

The basic elements of the pacemaker system [62, 73] are:

1. **Leads:** One or more flexible coiled metal wire normally two, that transmits electrical signals between the heart and the pacemaker. Each pacemaker lead is classified by its configuration: either one ("unipolar") or two ("bipolar") separated points of electrical contact with the heart.

2. **The Pacemaker Generator:** The pacemaker is both the power source and the brain of the pacing and sensing systems. It contains an implanted battery and controller as an electronic circuitry.

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

4. **Accelerometer:** It is an electromechanical device inside the pacemaker that measures the body motion or acceleration of motion of a body in order to allow modulated pacing.

In the double electrode pacemaker, the electrode is attached to the right atrium or the right ventricle. It has several operational modes that regulate the heart functioning. The specification document [68] describes all possible operating modes that are controlled by the different programmable parameters of the pacemaker. All the programmable parameters are related to real-time and action-reaction constraints, that is used to regulate the heart rate.

In order to understand the "language" of pacing, it is necessary to comprehend the coding system that produced by a combined working party of the North American Society of Pacing and Electrophysiology (NASPE) and the British Pacing and Electrophysiology Group (BPEG) known as NBG(NASPE/BPEG

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

Table-1 Bradycardia operating modes of pacemaker system

generic) code [88]. This is a code of five letters of which the first three are most often used. The code provides a description of the pacemaker pacing and sensing functions. The sequence is referred to as "bradycardia operating modes"(see Table-1). In practice, only the first three or four-letter positions are commonly used to describe bradycardia pacing functions. The first letter of the code indicates which chambers are being paced, the second letter indicates which chambers are being sensed, the third letter of the code indicates the response to sensing and the final letter, which is optional indicates the presence of rate modulation in response to the physical activity measured by the accelerometer. Accelerometer is an additional sensor in the pacemaker system that detects a physiological result of exercise or emotion and increase the pacemaker rate on the basis of a programmable algorithms. "X" is a wildcard used to denote any letter (i.e. "O", "A", "V" or "D"). $Triggered$ (T) refers to deliver a pacing stimulus and $Inhibited$ (I) refers to an inhibition from further pacing after sensing of an intrinsic activity from the heart chamber.The NBG code also uses a fifth letter relating to antitachycardia function which is not discussed in this report.

5.3 The modelling framework

Here, we will summarize the concepts of the EVENT B modelling language developed by Abrial [69, 59] and will indicate the links with the tool called RODIN [90]. The modelling process deals with various languages, as seen by considering the triptych of Bjoerner [63, 64, 65, 67]: $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$. Here, the domain $\mathcal{D}$ deals with properties, axioms, sets, constants, functions, relations, and theories. The system model $\mathcal{S}$ expresses a model or a refinement-based chain of models of the system. Finally, $\mathcal{R}$ expresses requirements for the system to be designed. Considering the EVENT B modelling language, we notice that the language can express *safety* properties, which are either *invariants* or *theorems* in a machine corresponding to the system. Recall that two main structures are available in EVENT B :

- Contexts express static information about the model.
- Machines express dynamic information about the model, invariants, safety properties, and events.

A EVENT B model is defining either a context or a machine. The triptych of Bjoerner [63, 64, 65, 67] $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$ is translated as follows: $\mathcal{C}, \mathcal{M} \longrightarrow \mathcal{R}$, where $\mathcal{C}$ is a context, $\mathcal{M}$ is a machine and $\mathcal{R}$ are the requirements. The relation $\longrightarrow$ is defined to be a satisfaction relation with respect to an underlying logico-mathematical theory. This satisfaction relation is supported by the RODIN platform. A machine is organizing events modifying state variables and it uses static informations defined in a context. These basic structure mechanisms are extended by the refinement mechanism which provides a mechanism for relating an abstract model and a concrete model by adding new events or by adding new variables. This mechanism allows us to develop gradually EVENT B models and to validate each decision step using the proof tool. The refinement relationship should be expressed as follows: a model M is refined by a model P, when P is simulating M. The final concrete model is close to the behaviour of real system that is executing events using real source code. We give details now on the definition of events, refinement and guidelines for developing complex system models.

5.3.1 Modelling actions over states

The event-driven approach [58, 69, 59] is based on the B notation. It extends the methodological scope of basic concepts to take into account the idea of *formal models*. Briefly, a formal model is characterized by a (finite) list x of *state variables* possibly modified by a (finite) list of *events*, where an invariant $I(x)$ states properties that must always be satisfied by the variables x and *maintained* by the activation of the events.

In the following, we summarize the definitions and principles of formal models and explain how they can be managed by tools [60, 71, 90].
Generalized substitutions are borrowed from the B notation. They provide a means to express changes to state variable values. In its simple form $x := E(x)$, a generalized substitution looks like an assignment statement. In this construct, x denotes a vector built on the set of state variables of the model, and $E(x)$ denotes a vector of expressions. Here, however, the interpretation we shall give to this statement is not that of an assignment statement. We interpret it as a *logical simultaneous substitution* of each variable of the vector x by the corresponding expression of the vector $E(x)$. There exists a more general normal form of this, denoted by the construct $x : |(P(x, x'))$. This should be read as *x is modified in such a way that the value of x afterwards, denoted by x', satisfies the predicate* $P(x, x')$, where x' denotes the *new value* of the vector and x denotes its *old value*. This is clearly nondeterministic in general.
An event has two main parts, namely, a *guard*, which is a predicate built on the state variables, and an *action*, which is a generalized substitution. An event can take one of three normal forms. The first form (BEGIN $x : |(P(x, x')$ END) shows an event that is not guarded, being therefore always enabled and semantically defined by $P(x, x')$. The second form (WHEN $G(x)$ THEN $x : |(Q(x, x'))$ END) and third form (ANY t WHERE $G(t, x)$ THEN $x : |(R(x, x', t))$ END) are guarded by a guard that states the necessary condition for these events to occur. The guard is represented by WHEN $G(x)$ in the second form, and by ANY t WHERE $G(t, x)$ (for $\exists t \cdot G(t, x)$) in the third form. We note that the third form defines a possibly nondeterministic event where t represents a vector of distinct local variables. The *before–after* predicate $BA(x, x')$, associated with each of the three event types, describes the event as a logical predicate expressing the relationship linking the values of the state variables just before (x) and just after (x') the *execution* of event EVENT evt. The second and the third forms are semantically equivalent to $G(x) \land Q(x, x')$ resp. $\exists t \cdot (G(t, x) \land R(x, x', t)$. Table-2 summarizes the three possible forms for writing a B event:
Proof obligations (INV 1 and INV 2) are produced by the RODIN tool [90] from events to state that an invariant condition $I(x)$ is preserved. Their general form follows immediately from the definition of the before–after predicate $BA(e)(x, x')$ of each event e (see Table-2). Note that it follows from the two guarded forms of the events that this obligation is trivially discharged when the guard of the event is false. Whenever this is the case, the event is said to be *disabled*. The proof obligation FIS expresses the feasibility of the event e with respect to the invariant I.

5.3.2 Model refinement

The refinement of a formal model allows us to enrich the model via a *step-by-step* approach and is the foundation of our correct-by-construction approach [81]. Refinement provides a way to strengthen invariants and to add details to a model. It is also used to transform an abstract model to a more concrete version by modifying the state description. This is done by extending the list of state variables (possibly suppressing some of them), by refining each abstract event to a corresponding concrete version, and by adding new events. The abstract (x) and concrete (y) state variables are linked by means of a *gluing invariant* $J(x, y)$. A number of proof obligations ensure that (1) each abstract event is correctly refined by its corresponding concrete version, (2) each new event refines $skip$, (3) no new event takes control for ever, and (4) relative deadlock freedom is preserved. Details of the formulation of these proofs follows.
We suppose that an abstract model AM with variables x and invariant $I(x)$ is refined by a concrete model CM with variables y and gluing invariant $J(x, y)$. If $BA(e)(x, x')$ and $BA(f)(y, y')$ are respectively the abstract and concrete before–after predicates of the same event, e and f respectively, we have to prove the following statement, corresponding to proof obligation (1):

$$I(x) \land J(x, y) \land BA(f)(y, y') \Rightarrow \exists x' \cdot (BA(e)(x, x') \land J(x', y'))$$

Now, proof obligation (2) states that $BA(f)(y, y')$ must refine $skip$ ($x' = x$), generating the following simple statement to prove (2).

$$I(x) \land J(x, y) \land BA(f)(y, y') \Rightarrow J(x, y')$$

In refining a model, an existing event can be refined by strengthening the guard and/or the before–after predicate (effectively reducing the degree of nondeterminism), or a new event can be added to refine the

Event e	Before-after Predicate BA(e)(x,x')
BEGIN $x : \|(P(x,x'))$ END	$P(x,x')$
WHEN $G(x)$ THEN $x : \|(Q(x,x'))$ END	$G(x) \ \wedge \ Q(x,x')$
ANY t WHERE $G(t,x)$ THEN $x : \|(R(x,x',t))$ END	$\exists t \cdot (\, G(t,x) \ \wedge \ R(x,x',t)\,)$

PROOF OBLIGATIONS

- (INV1) $Init(x) \ \Rightarrow \ I(x)$
- (INV2) $I(x) \ \wedge \ BA(e)(x,x') \ \Rightarrow \ I(x')$
- (FIS) $I(x) \ \wedge \ \mathsf{grd}(e)(x) \ \Rightarrow \exists y.BA(e)(x,y)$

Table-2 EVENT B events and proof obligations

skip event. The feasibility condition is crucial to avoiding possible states that have no successor, such as division by zero. Furthermore, this refinement guarantees that the set of traces of the refined model contains (up to stuttering) the traces of the resulting model. The refinement of an event e by an event f means that the event f simulates the event e.

The EVENT B modelling language is supported by the RODIN platform [90] and has been introduced in publications [59, 69], where there are many case studies and discussions about the language itself and the foundations of the EVENT B approach. The language of *generalized substitutions* is very rich, enabling the expression of any relation between states in a set-theoretical context. The expressive power of the language leads to a requirement for help in writing relational specifications, which is why we should provide guidelines for assisting the development of EVENT B models.

5.3.3 Guidelines for EVENT B Modelling

Considering design patterns [74], the purpose is to capture structures and to make decisions within a design that are common to similar modeling and analysis tasks. They can be re-applied when undertaking similar tasks in order to reduce the duplication of effort. The design pattern approach is the possibility to reuse solutions from earlier developments in the current project. This will lead to a *correct refinement* in the chain of models, without producing proof obligations. Since the correctness (i.e proof obligations are proved) of the pattern has been proved during its development, nothing is to be proved again when using this pattern.

The pacemaker systems are characterized by their functions, which can be expressed by analyzing *action-reaction* and *real time* patterns. Sequences of inputs are recognized, and outputs can be emitted in response within a fixed time interval. So, the most common elements in pacemaker system are bounded time interval for every action, reaction and action-reaction pair. The action-reaction within a time limit can be viewed as an abstraction of the pacemaker system. We recognize the following two design patterns when modeling this kind of system according to the relationship between the action and corresponding reaction.

Under action-reaction chapter [59] two basic types of design patterns are,

Action and Weak Reaction: Once an action emits, a reaction should start in response. For a quick instance, if an action stops, the reaction should follow. Sometimes reaction does not change immediately according to the action because the action moves too quickly (the continuance of an action is too short, or the interval between actions is too short). This is known as pattern of action and weak reaction.

Action and Strong Reaction: For every action, there is a corresponding reaction. To keep proper synchronization between action and corresponding reaction, known as pattern of action and strong reaction.

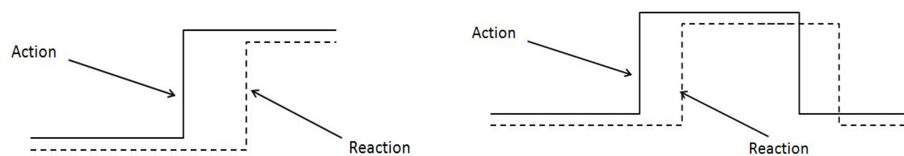


Fig. 3 Action-Reaction Patterns

The action-reaction events of a pacemaker system are based on the time constraint pattern in IEEE 1394 proposed by Cansell et. al and on the 2-Slots Simpson Algorithm case studies [70, 91]. This time pattern is fully based on timed automaton. The timed automaton is a finite state machine that is useful to model components of real-time systems. In a model, timed automata interacts with each other and defines a timed transition system. Besides ordinary action transitions that can represent input, output and internal actions. A timed transition system has time progress transitions. Such time progress transitions result in synchronous progress of all clock variables in the model. Here we apply the time pattern in modeling to synchronize the sensing and pacing stimulus functions of the pacemaker system in continuous progressive time constraint. In the model, events are controled under time constraints, which means action of any event activates only when time constraint satisfies on specific time. The time progress is also an event, so there is no modification of the underlying EVENT B language. It is only a modeling technique instead of a specialized formal system. The timed variable is in $\mathbb{N}$ $(natural\ numbers)$ but time constraint can be written in terms involving unknown constants or expressions between different times. Finally, the timed event observations can be constrained by other events which determine future activations.

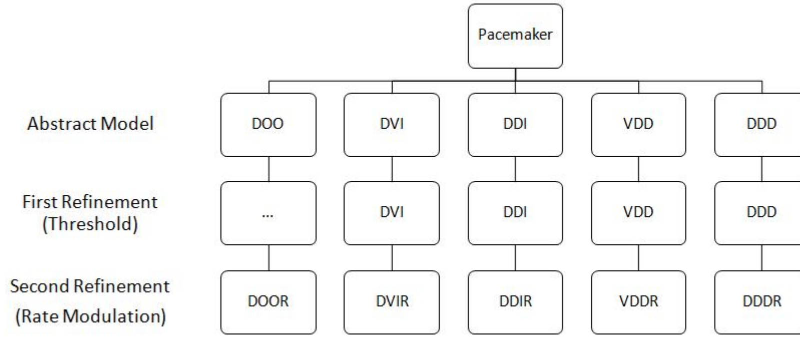


Fig. 4 Refinement structure of the double electrode pacemaker operating modes

5.4 Formal Development

The two electrode pacemaker system is more complex than one electrode pacemaker system. We have designed the block diagram (see Fig. 4) of hierarchical tree structure of the possible operating modes of the double electrode pacemaker. The hierarchical tree structure shows the stepwise refinement from abstract to concrete model. Each level of refinement introduces the new features of pacemaker as functional and parametric requirements. The root of this tree indicates the double electrode pacemaker. In a double electrode pacemaker, two electrodes are placed in both chambers; atrium and ventricular. These atrium and ventricular are the right atrium and ventricular. The next five branches of tree show the five opearting modes; DOO, DVI, DDI, VDD, and DDD (see Table-1). In thses operating modes, the pacemaker uses the both electrodes to pace in both chambers synchronously. It is an abstract level of the model. In the abstract model, we introduce all the operating modes abstractly with required properties of the pacemaker. From first refinement to last refinement, there is only one branch in every operating modes of the atrium and the ventricular chambers. The subsequent refinement models introduce all detailed information for the resulting system. Every refinement level shows an extension of previous operating modes as an introduction of a new feature or functional requirement. the triple dots (...) represents that there is no refinement at that level in particular operating mode (DOO). In abstract level and first refinement level, we have similar operating modes. But in the second refinement level, we have achieved the additional rate adaptive operating modes(i.e.DOOR, DVIR, DDIR, VDDR and DDDR). These operating modes are different from the previous levels operating modes. These refinement structure is very helpful to model the functional requirements of the double electrode cardiac pacemaker. The following outline is given about every refinement level to understand the basic notion of the cardiac pacemaker model:-

- **Abstract Model :** Specifies the pacing, sensing and timing components with the help of action-reaction and real-time pattern using some initial events ($Actuator_ON_A$, $Actuator_OFF_A$, $Actuator_ON_V$, $Actuator_OFF_V$,
$Sensor_ON_A$, $Sensor_OFF_A$, $Sensor_ON_V$, $Sensor_OFF_V$, tic, tic_AV).
- **Refinement 1 :** Introduces the $threshold$ parameter to filter the exact sensing value within a sensing period to control the sensing and the pacing event and introduces more invariants to satisfy the pacing and sensing requirements of the system in both chambers.
- **Refinement 2 :** Introduces the $accelerometer\ sensor$ component and $rate\ modulation$ function to achieve the new rate adaptive operating modes of the pacemaker.

We have presented here only selected parts of our formalization and omit proof details.

5.4.1 The Context and Initial Model

In this section we describe the formal development of initial modes of double electrode cardiac pacemaker system. In the abstract model, we introduce the basic notions of action-reaction and real-time constraints using actuator and sensor in different heart chambers. In this abstraction, we begin with an abstract model

of a double electrode cardiac pacemaker system focusing on pacing and sensing modes properties and operations control the pumping rate of natural pacemaker or human heart. However, some pacing modes related to rate modulation are not described in this level. Instead they will emerged into final refinement. Thus, in this level, for every modes of pacemaker are treated in the same way as common basic modes, which are essential for the double electrode cardiac pacemaker. The model consists of several modules, each corresponding to an operating mode of the pacemaker. Here, we define the required context and then abstraction of the double electrode cardiac pacemaker system.

We begin by defining the Event-B context. The context uses the sets and constants to define the axioms and theorems. The axioms and theorems represent the logical formulation of the system. The logical formulation is the constant behaviour and the set of properties of the system. In the context, we define the constants LRL and URL that relate to the lower rate limit (LRL) (minimum number of pace pulses delivered per minute by pacemaker) and upper rate limit (URL) (how fast the pacemaker will allow the heart to be paced). These constants are extracted from the pacemaker specification document [68]. The lower rate limit (LRL) must be between 30 and 175 pulse per minute (ppm) and upper rate limit (URL) must be between 50 and 175 pulse per minute (ppm). To test the model by ProB model checker [89], we have taken a nominal value of lower rate limit (LRL) as 60 ppm and upper rate limit (URL) as 120 ppm according the pacemaker specification document [68].

The two new constants URI (upper rate interval) and LRI (lower rate interval) are defined by axioms ($axm3$ and $axm4$). The pacemaker (or pacing) rate is programmed in milliseconds. To convert a heart rate from beats per minute (bpm) to milliseconds, divide 60,000 by the heart rate. For example, a heart rate of 70 bpm equals 857 milliseconds. Additionaly, we define an enumerated set $status$ of an electrode as ON and OFF states and new constant atrioventricular interval $FixedAV$ in $axm5$ and $axm6$, respectively. Refractory period constants Atria Refractory Period ARP, Ventricular Refractory Period VRP and Post Ventricular Atria Refractory Period $PVARP$ are defined in $axm7$,$axm8$ and $axm9$, respectively. Another new constant V_Blank is defined as blanking period as initial period of VRP. Finally, we have introduced some basic initial properties between defined constants of the system by axioms($axm11$, $axm12$, , $axm13$, $axm14$ and $axm15$).

$$
\begin{array}{l}
axm1 : LRL \in 30 \,..\, 175 \\
axm2 : URL \in 50 \,..\, 175 \\
axm3 : URI \in \mathbb{N}_1 \land URI = 60000/URL \\
axm4 : LRI \in \mathbb{N}_1 \land LRI = 60000/LRL \\
axm5 : status = \{ON, OFF\} \\
axm6 : FixedAV \in 70 \,..\, 300 \\
axm7 : ARP \in 150 \,..\, 500 \\
axm8 : VRP \in 150 \,..\, 500 \\
axm9 : PVARP \in 150 \,..\, 500 \\
axm10 : V_Blank \in 30 \,..\, 60 \\
axm11 : LRL < URL \\
axm12 : URI < LRI \\
axm13 : URI > PVARP \\
axm14 : URI > VRP \\
axm15 : VRP \geq PVARP
\end{array}
$$

Abstraction of DOO mode:

In the double electrode pacemaker system, the pacemaker delivers a pacing stimulus in the atrial and ventricular chambers. In DOO operating mode of double electrode cardiac pacemaker system, the first letter 'D' represents that the pacemaker paces both atrial and ventricle, second letter 'O' represents that the pacemaker does not sense the atrial and the ventricle chambers and final letter 'O' represents that there is no any inhibits or triggers in both chambers (atrail and ventricular). In the block diagram (Fig-5) of heart pacing in DOO operating mode pacemaker will pace both chambers (atrial and ventricular) asynchronously at the constant rate regardless of the underlying rhythm. It does not sense, If the native rhythm is slower than the constant rate then atrial and ventricular capture will most likely be seen at the constant rate.

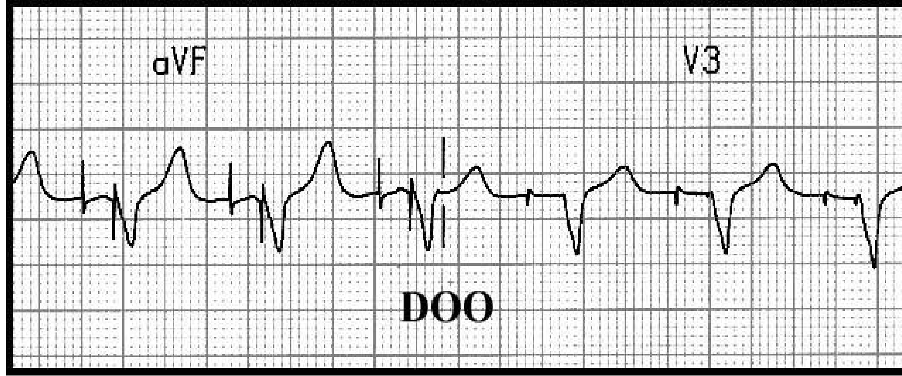


Fig. 5 Basic block diagram of ECG rhythm strip in DOO Operating Mode

In our initial model, we have formalized the functional behaviors of the cardiac pacemaker system , where two new variables $PM_Actuator_A$ and $PM_Actuator_V$ are represented ON or OFF states of the pacemaker's actuators for pacing in the atrial and ventricular chambers. An interval between two paces is defined by a new variable $Pace_Int$ that must be between upper rate interval (URI) and lower rate interval (LRI), is represented by an invariant ($inv3$). The variable $Pace_Int$ is an interval between two paces of ventricular chamber that is initialized by the system before start the pacing. This interval is equal to atrioventricular(AV) interval plus ventriculoatrial(VA) interval. A variable sp (since pace) represents a current *clock counter*. A variable $last_sp$ represents the last interval (in ms.) between two paces and a safety property in invariant ($inv5$) states that last interval must be between URI and LRI. In invariant ($inv6$) a new variable $Atria_state$ is used as boolean type to control the state of the atrial chamber. The invariant ($inv7$) states that the pacemaker's actuator of atrial and ventricular chambers are OFF when clock counter sp is less than ventriculoatrial (VA) interval and atrial state ($Atria_state$) is $FALSE$. The next invariant ($inv8$) represents that the pacemaker's actuator of both chambers are OFF when clock counter sp is greater than atrioventricular (AV) interval and atrial state ($Atria_state$) is $TRUE$. The last invariants ($inv9$ and $inv10$) state that pacemaker's actuator of atrial is ON when clock counter sp is equal to ventriculoatrial (VA) interval $Pace_Int - FixedAV$ and pacemaker's actuator of ventricular is ON when clock counter sp is equal to the pace interval $Pace_Int$, respectively.

$$
\begin{array}{l}
inv1 : PM_Actuator_A \in status \\
inv2 : PM_Actuator_V \in status \\
inv3 : Pace_Int \in URI \,..\, LRI \\
inv4 : sp \in 1 \,..\, Pace_Int \\
inv5 : last_sp \geq URI \land last_sp \leq LRI \\
inv6 : Atria_state \in BOOL \\
inv7 : sp < (Pace_Int - FixedAV) \land Atria_state = FALSE \\
\qquad \Rightarrow \\
\qquad PM_Actuator_V = OFF \land PM_Actuator_A = OFF \\
inv8 : sp > (Pace_Int - FixedAV) \land sp < Pace_Int \land \\
\qquad Atria_state = TRUE \Rightarrow \\
\qquad PM_Actuator_A = OFF \land PM_Actuator_V = OFF \\
inv9 : PM_Actuator_A = ON \\
\qquad \Rightarrow \\
\qquad sp = Pace_Int - FixedAV \\
inv10 : PM_Actuator_V = ON \Rightarrow sp = Pace_Int
\end{array}
$$

In the abstract specification of DOO operating mode, there are five events $Pace_ON_A$ to start pacing in atrial, $Pace_OFF_A$ to stop pacing in atrial, $Pace_ON_V$ to start pacing in ventricular, $Pace_OFF_V$ to stop pacing in ventricular and tic to increment the current clock counter sp under real time constraints.

EVENT Pace_ON_A
WHEN
grd1 : $PM_Actuator_A = OFF$
grd2 : $Atria_state = FALSE$
grd3 : $sp = Pace_Int - FixedAV$
THEN
act1 : $PM_Actuator_A := ON$
END

The events $Pace_ON_A$ and $Pace_OFF_A$ start and stop the pulse discharging into the atrial chamber. The guards and an action of event $(Pace_ON_A)$ state that pacemaker's actuator $(PM_Actuator_A)$ of atrial is ON when pacemaker's actuator $(PM_Actuator_A)$ of atrial is OFF, atrial state $(Atria_state)$ is FALSE and clock counter sp is equal to ventriculoatrial (VA) interval $(Pace_Int - FixedAV)$.

EVENT Pace_OFF_A
WHEN
grd1 : $PM_Actuator_A = ON$
grd2 : $PM_Actuator_V = OFF$
grd3 : $sp = Pace_Int - FixedAV$
THEN
act1 : $PM_Actuator_A := OFF$
act2 : $Atria_state := TRUE$
END

The guards and actions of event $(Pace_OFF_A)$ state that pacemaker's actuator $(PM_Actuator_A)$ of atrial chamber is OFF and atrial state $(Atria_state)$ is TRUE, when pacemaker's actuator $(PM_Actuator_A)$ of atrial is ON, pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF and clock counter sp is equal to ventriculoatrial (VA) interval $(Pace_Int - FixedAV)$.

EVENT Pace_ON_V
WHEN
grd1 : $PM_Actuator_V = OFF$
grd2 : $PM_Actuator_A = OFF$
grd3 : $sp = Pace_Int$
THEN
act1 : $PM_Actuator_V := ON$
act2 : $last_sp := sp$
END

The events $Pace_ON_V$ and $Pace_OFF_V$ also synchronize start and stop the pulse discharging into the ventricular chamber. The guards and actions of event $(Pace_ON_V)$ state that pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and clock counter sp assigns to a variable $(last_sp)$ when pacemaker's actuator of both chambers $(PM_Actuator_A$, $PM_Actuator_V)$ is OFF and and clock counter sp is equal to the pace interval $(Pace_Int)$.

EVENT Pace_OFF_V
WHEN
grd1 : $PM_Actuator_V = ON$
grd2 : $PM_Actuator_A = OFF$
grd3 : $Atria_state = TRUE$
grd4 : $sp = Pace_Int$
THEN
act1 : $PM_Actuator_V := OFF$
act2 : $sp := 1$
act3 : $Atria_state := FALSE$
END

The guards and actions of event $(Pace_OFF_V)$ state that pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF, clock counter sp resets to 1 and atrial state $(Atria_state)$ sets into TRUE state when pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON, pacemaker's actuator $(PM_Actuator_A)$ of atrial is OFF, atrial state $(Atria_state)$ is TRUE and clock counter sp is equal to the pace interval $(Pace_Int)$.

EVENT tic
WHEN
grd1 : $(sp < (Pace_Int - FixedAV))$
$\vee$
$(sp \geq (Pace_Int - FixedAV) \wedge sp < Pace_Int \wedge$
$Atria_state = TRUE \wedge PM_Actuator_V = OFF$
$\wedge PM_Actuator_A = OFF)$
THEN
act1 : $sp := sp + 1$
END

The last event tic of this abstraction progressively increases the current clock counter sp under pre-defined pace interval $(Pace_Int)$. The guard of this event controls the pacing stimulus into the heart chambers (atrial and ventricular) and synchronizes ON and OFF states of pacemaker's actuator of each chamber (atrial and ventricular) under real time constraints. The guards of this event provides the required conditions to increase the current clock counter sp by 1 (ms.).

Abstraction of DVI mode:

In DVI operating mode of double electrode cardiac pacemaker system, the first letter 'D' represents that the pacemaker paces both atrial and ventricle, second letter 'V' represents that the pacemaker senses the ventricle only and final letter 'I' represents that the ventricular sensing inhibits atrial and ventricular pacing. In the block diagram (Fig-6) of heart pacing in DVI operating mode a ventriculoatrial (VA) interval follows the timing for each atrioventricular (AV) interval and an atrioventricular (AV) interval follows the timing for each ventriculoatrial (VA) interval, except with an R wave sensed during the ventriculoatrial (VA) interval that starts timing of a new ventriculoatrial (VA) interval.

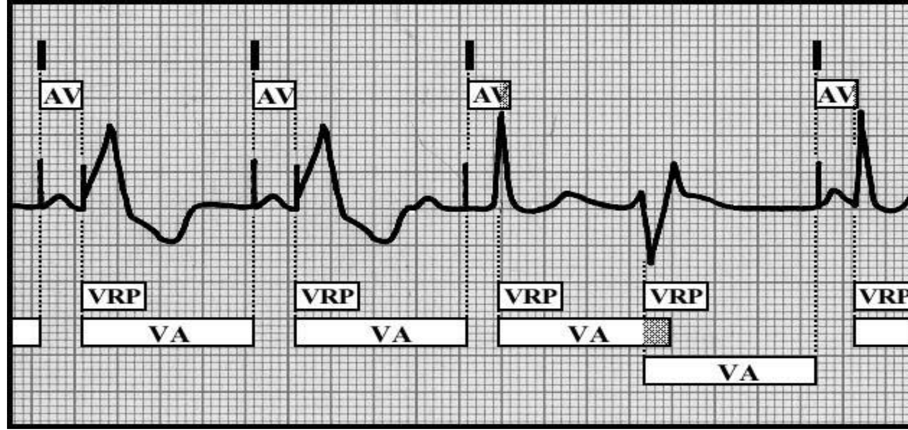


Fig. 6 Basic block diagram of ECG rhythm strip in DVI Operating Mode

In the abstract model of DVI mode, two new variables $(PM_Actuator_A)$ and $(PM_Actuator_V)$ represent the presence (ON) or absence (OFF) of pulse in atrial and ventricular chambers, respectively. The variable (PM_Sensor_V) also represents the presence (ON) or absence (OFF) of pacemaker's sensor in ventricular chamber. An interval between two paces is defined by a new variable $Pace_Int$ that must be between upper rate interval (URI) and lower rate interval (LRI), is represented by an invariant $(inv3)$. The variable $Pace_Int$ is an interval between two paces of ventricular chamber that is initialized by the system before start the pacing. This interval is equal to atrioventricular(AV) interval plus ventriculoatrial(VA) interval. A variable sp (since pace) represents the current *clock counter*. A variable $last_sp$ represents the last interval (in ms.) between two paces and a safety property in invariant $(inv6)$ states that last interval must be greater than or equal to VRP and less than and equal to LRI. In invariant $(inv7)$ a new variable AV_Count_STATE is used as boolean type to control the counting interval of atrioventricular (AV) interval. The variable (AV_Count) define as natural number to count the atrioventricular (AV) interval. A invariant $(inv9)$ represents that the safety property and states that during the ventricular refractory period (VRP), pacemaker's actuator $(PM_Actuator_A$, $PM_Actuator_V)$ of atrial and ventricular chambers are OFF and pacemaker's sensor (PM_Sensor_V) of ventricular chamber is also OFF. The last invariants $(inv10$ and $inv11)$ state that pacemaker's actuator of atria is ON when clock counter sp is greater than or equal to ventriculoatrial (VA) interval $Pace_Int - FixedAV$ and pacemaker's actuator of ventricular is ON when clock counter sp is equal to the pace interval $Pace_Int$, respectively.

$$
\begin{array}{l}
inv1 : PM_Actuator_A \in status \\
inv2 : PM_Actuator_V \in status \\
inv3 : PM_Sensor_V \in status \\
inv4 : Pace_Int \in URI \,..\, LRI \\
inv5 : sp \in 1 \,..\, Pace_Int \\
inv6 : last_sp \geq VRP \land last_sp \leq LRI \\
inv7 : AV_Count_STATE \in BOOL \\
inv8 : AV_Count \in \mathbb{N} \\
inv9 : sp < VRP \Rightarrow PM_Actuator_A = OFF \land \\
\qquad\quad PM_Actuator_V = OFF \land PM_Sensor_V = OFF \\
inv10 : PM_Actuator_A = ON \Rightarrow \\
\qquad\quad sp \geq Pace_Int - FixedAV \\
inv11 : PM_Actuator_V = ON \Rightarrow sp = Pace_Int
\end{array}
$$

In the abstract specification of DVI operating mode, there are eight events $Actuator_ON_A$ to start pacing in atria, $Actuator_OFF_A$ to stop pacing in atria, $Actuator_ON_V$ to start pacing in ventricular, $Actuator_OFF_V$ to stop pacing in ventricular, $Sensor_ON_V$ to star sensing in ventricular, $Sensor_OFF_V$ to stop sensing in ventricular, tic to increment the current clock counter sp under real time constraints and tic_AV to count the atrioventricular (AV) interval.

EVENT Actuator_ON_V
WHEN
grd1 : $PM_Actuator_V = OFF$
grd2 : $(sp = Pace_Int)$
grd3 : $sp \geq VRP$
THEN
act1 : $PM_Actuator_V := ON$
act2 : $last_sp := sp$
END

The events $Actuator_ON_V$ and $Actuator_OFF_V$ start and stop the pacemaker's actuator in ventricular chamber and synchronizes ON and OFF states. The guards of event $Actuator_ON_V$ represent that when pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF, current clock counter sp is equal to pace interval $(Pace_Int)$ and greater than or equal to VRP. The actions represent that if all guards are satisfy then the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and clock counter sp assigns to a new variable $(last_sp)$.

EVENT Actuator_OFF_V
WHEN
grd1 : $PM_Actuator_V = ON$
grd2 : $(sp = Pace_Int)$
grd3 : $AV_Count \geq FixedAV$
THEN
act1 : $PM_Actuator_V := OFF$
act2 : $AV_Count := 0$
act3 : $AV_Count_STATE := FALSE$
act4 : $PM_Sensor_V := OFF$
act5 : $sp := 1$
act6 : $PM_Actuator_A := OFF$
END

The guards of event $Actuator_OFF_V$ states that pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and clock counter sp is equal to pace interval $(Pace_Int)$ and atrioventricular (AV) counter (AV_Count) is greater than or equal to atrioventricular (AV) interval $(FixedAV)$. The actions of this event reset all the required parameters of cardiac pacemaker system. The actions $(act1 - act6)$ of this event state that the pacemaker's actuator $(PM_Actuator_V)$ of ventricular sets in OFF state, assigns the value of variable (AV_count) as 0, atrioventricular (AV) counter state (AV_Count_STATE) sets in FALSE state, pacemaker's sensor (PM_Sensor_V) of ventricular sets in OFF state, assigns the value of clock counter sp as 1 and sets in OFF state of pacemaker's actuator $(PM_Actuator_A)$ of atrial.

EVENT Actuator_ON_A
WHEN
grd1 : $PM_Sensor_V = ON$
grd2 : $sp \geq Pace_Int - FixedAV \wedge sp \geq VRP \wedge sp < Pace_Int$
grd3 : $PM_Actuator_A = OFF$
grd4 : $AV_Count_STATE = FALSE$
THEN
act1 : $PM_Actuator_A := ON$
act2 : $PM_Sensor_V := OFF$
END

The actions $(act1, act2)$ of event $(Actuator_ON_A)$ state that the pacemaker's actuator $(PM_Actuator_A)$ of atria sets in ON state and pacemaker's sensor (PM_Sensor_V) of ventricular sets in OFF state when all guards satisfy. The first guard of this event states that pacemaker's sensor (PM_Sensor_V) of ventricular

is ON, the next guard $(grd2)$ states that clock counter sp is greater than or equal to ventriculoatrial (VA) interval, VRP and less than pace interval $(Pace_Int)$), the third guard shows that the pacemaker's actuator $(PM_Actuator_A)$ of atrial is OFF and and in last guard states that atrioventricular (AV) counter state (AV_Count_STATE) is FALSE.

EVENT Actuator_OFF_A
  WHEN
    grd1 : $PM_Actuator_A = ON$
    grd2 : $sp \geq Pace_Int - FixedAV \wedge sp \geq VRP \wedge sp < Pace_Int$
    grd3 : $AV_Count_STATE = FALSE$
  THEN
    act1 : $PM_Actuator_A := OFF$
    act2 : $AV_Count_STATE := TRUE$
  END

The actions $(act1, act2)$ of event $(Actuator_OFF_A)$ state that pacemaker's actuator $(PM_Actuator_A)$ of atria is OFF and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE. The guards $(grd1 - grd2)$ of this event state that pacemaker's actuator $(PM_Actuator_A)$ of atria is ON, clock counter sp is greater than or equal to ventriculoatrial (VA) interval, VRP and less than pace interval $(Pace_Int)$. The last guard shows that atrioventricular (AV) counter state (AV_Count_STATE) is FALSE.

EVENT Sensor_ON_V
  WHEN
    grd1 : $PM_Sensor_V = OFF$
    grd2 : $(sp < Pace_Int - FixedAV)$
    $\vee$
    $(sp \geq Pace_Int - FixedAV \wedge AV_Count_STATE = TRUE)$
    grd3 : $sp \geq VRP$
  THEN
    act1 : $PM_Sensor_V := ON$
  END

The events $(Sensor_ON_V$ and $Sensor_OFF_V)$ is used to control the sensing activities from the ventricular chamber. The pacemaker's sensor (PM_Sensor_V) of ventricular chamber synchronizes the ON and OFF states under real time constraints. The guard $(grd1)$ of event $(Sensor_ON_V)$ represents that if pacemaker's sensor (PM_Sensor_V) of ventricular is OFF and a guard $(grd2)$ represents that current clock counter sp is less than ventriculoatrial (VA) interval or greater than or equal to ventriculoatrial (VA) interval and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE. The last guard $(grd3)$ represents that current clock counter sp is greater than or equal to VRP. If all guards are true then in action part of this event pacemaker's sensor (PM_Sensor_V) of ventricular is ON.

EVENT Sensor_OFF_V
  WHEN
    grd1 : $PM_Sensor_V = ON$
    grd2 : $sp \geq VRP \wedge sp < Pace_Int$
  THEN
    act1 : $PM_Sensor_V := OFF$
    act2 : $AV_Count := 0$
    act3 : $AV_Count_STATE := FALSE$
    act4 : $last_sp := sp$
    act5 : $sp := 1$
    act6 : $PM_Actuator_V := OFF$
    act7 : $PM_Actuator_A := OFF$
  END

The event $(Sensor_OFF_V)$ is used to set the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF. The guards of this event represent that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON and current clock counter sp is greater than or equal to VRP and less than pace interval $(Pace_Int)$. All actions of this event is same as event $(Actuator_OFF_V)$ of DOO operating mode, which are already described. Here only extra action $(act1)$ is added to set the pacemaker's sensor (PM_Sensor_V) of ventricular is in OFF state.

EVENT tic
WHEN
grd1 : $(sp < VRP \wedge PM_Sensor_V = OFF)$
$\vee$
$(sp \geq VRP \wedge sp < Pace_Int \wedge PM_Sensor_V = ON \wedge$
$AV_Count_STATE = FALSE)$
THEN
act1 : $sp := sp + 1$
END

The event (tic) of this abstraction progressively increases the current clock counter sp under pre-defined pace interval $(Pace_Int)$. The guard $(grd1)$ of this event control the pacing stimulus into the heart chambers (atria and ventricular), synchronizes ON and OFF states of the pacemaker's actuator $(PM_Actuator_A, PM_Actuator_V)$ of each chamber (atria and ventricular) and also control the sensing intrinsic stimulus of ventricular chamber and synchronizes the ON and OFF states of ventricular pacemaker's sensor (PM_Sensor_V) under real time constraints.

EVENT tic_AV
WHEN
grd1 : $AV_Count \leq FixedAV$
grd2 : $AV_Count_STATE = TRUE$
grd3 : $sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int$
THEN
act1 : $AV_Count := AV_Count + 1$
act2 : $sp := sp + 1$
END

The last event (tic_AV) of this abstraction progressively counts the atrioventricular (AV) interval and also increases the current clock counter sp is represented in actions $(act1$ and $act2)$. The guards $(grd1 - grd3)$ of this event states that atrioventricular (AV) counter (AV_Count) is less than and equal to atrioventricular (AV) interval $(FixedAV)$, atrioventricular (AV) counter state (AV_count_STATE) is in TRUE state and current clock counter sp is within the atrioventricular (AV) interval.

Abstraction of DDI mode:

In DDI operating mode of double electrode pacemaker system, the first letter 'D' represents that the pacemaker paces both atrial and ventricle, second letter 'D' represents that the pacemaker senses both atrial and ventricle and final letter 'I' represents two conditional meaning that depends on atrial and ventricular sensing; first is that atrial sensing inhibits atrial pacing and does not trigger ventricular pacing and second is that ventricular sensing inhibits ventricular and atrial pacing. In the block diagram (Fig-7) of heart pacing in DDI operating mode a new LRI follows the timing of each preceding LRI. The timing of an atrioventricular (AV) interval occurs within this period only following a completed ventriculoatrial (VA) interval (i.e., when no atrial sensing occurs).

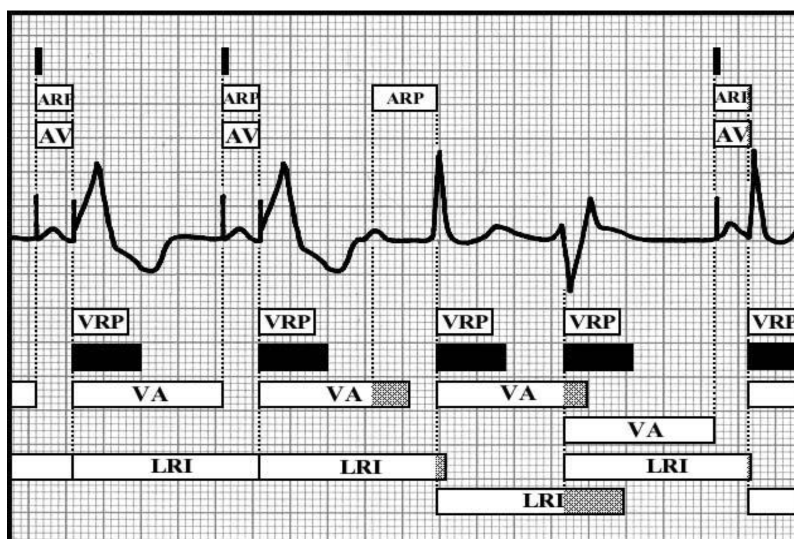


Fig. 7 Basic block diagram of ECG rhythm strip in DDI Operating Mode

In this abstract model, we have formalized a bradycardia operating mode *DDI* of the double electrode pacemaker. In this operating mode, the pacemaker uses actuators and sensors in both chambers. We have defined two new variables $PM_Actuator_A$ and $PM_Actuator_V$ that represent ON or OFF states of pacemaker's actuators for pacing in the atrial and ventricular chambers. Similarly next two variables PM_Sensor_A and PM_Sensor_V represent ON or OFF states of pacemaker's sensor for sensing an intrinsic pulse from both the atrial and ventricular chambers. An interval between two paces is defined by a new variable $Pace_Int$ that must be between upper rate interval (URI) and lower rate interval (LRI), is represented by an invariant ($inv5$). A variable sp (since pace) represents the current *clock counter*. A variable $last_sp$ represents the last interval (in ms.) between two paces and a safety property in invariant ($inv7$) states that last interval must be between PVARP and pace interval $Pace_Int$. Another new variable AV_Count_STATE is defined as boolean type to control the atrioventricular (AV) interval state and next variable AV_Count is defined as natural number to count the atrioventricular (AV) interval. Extra two new invariants ($inv11$ and $inv12$) represent the safety properties. The invariant ($inv11$) states that when clock counter sp is less than ventricular refractory period (VRP) and atrioventricular (AV) counter state is FALSE then the pacemaker's actuators and sensors of both chambers are OFF. The next invariant ($inv12$) represents that the pacemaker's actuator of ventricular is ON when clock counter sp is equal to the pace interval $Pace_Int$.

$$
\begin{array}{l}
inv1 : PM_Actuator_A \in status \\
inv2 : PM_Actuator_V \in status \\
inv3 : PM_Sensor_A \in status \\
inv4 : PM_Sensor_V \in status \\
inv5 : Pace_Int \in URI\ ..\ LRI \\
inv6 : sp \in 1\ ..\ Pace_Int \\
inv7 : last_sp \geq PVARP \land last_sp \leq Pace_Int \\
inv8 : AV_Count_STATE \in BOOL \\
inv9 : AV_Count \in \mathbb{N} \\
inv10 : Pace_Int - FixedAV < Pace_Int \\
inv11 : sp < VRP \land AV_Count_STATE = FALSE \\
\qquad \Rightarrow \\
\qquad PM_Actuator_A = OFF \land \\
\qquad PM_Actuator_V = OFF \land \\
\qquad PM_Sensor_A = OFF \land \\
\qquad PM_Sensor_V = OFF \\
inv12 : PM_Actuator_V = ON \Rightarrow sp = Pace_Int
\end{array}
$$

In the abstract specification of DDI operating mode, there are ten events $Actuator_ON_A$ to start pacing in atria, $Actuator_OFF_A$ to stop pacing in atria, $Actuator_ON_V$ to start pacing in ventricular,

$Actuator_OFF_V$ to stop pacing in ventricular, $Sensor_ON_V$ to start sensing in ventricular, $Sensor_OFF_V$ to stop sensing in ventricular, $Sensor_ON_A$ to star sensing in atrial, $Sensor_OFF_A$ to stop sensing in atrial, tic to increment the current clock counter sp under real time constraints and tic_AV to count the atrioventricular (AV) interval.

EVENT Actuator_ON_V
WHEN
grd1 : $PM_Actuator_V = OFF$
grd2 : $(sp = Pace_Int)$
grd3 : $sp \geq VRP \land sp \geq PVARP$
THEN
act1 : $PM_Actuator_V := ON$
act2 : $last_sp := sp$
END

The events $Actuator_ON_V$ and $Actuator_OFF_V$ start and stop the pacemaker's actuator $(PM_Actuator_V)$ in ventricular chamber and synchronizes ON and OFF states. The guards of event $(Actuator_ON_V)$ represent that when pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF, current clock counter sp is equal to pace interval $(Pace_Int)$, in next guard current clock counter sp is greater then or equal to ventricular refractory period (VRP) and post ventricular refractory period (PVARP). The actions of this event represent that if all guards are satisfy then the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and current clock counter sp assigns to a variable $(last_sp)$.

EVENT Actuator_OFF_V
WHEN
grd1 : $PM_Actuator_V = ON$
grd2 : $(sp \geq Pace_Int) \land sp \geq VRP \land sp \geq PVARP$
grd3 : $AV_Count_STATE = TRUE$
grd4 : $PM_Sensor_A = OFF$
grd5 : $PM_Actuator_A = OFF$
THEN
act1 : $PM_Actuator_V := OFF$
act2 : $AV_Count := 0$
act3 : $AV_Count_STATE := FALSE$
act4 : $PM_Sensor_V := OFF$
act5 : $sp := 1$
END

The guards $(grd1, grd2)$ of event $Actuator_OFF_V$ state that pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and current clock counter sp is greater than or equal to pace interval $(Pace_Int)$, VRP and PVARP. The third guard $(grd3)$ states that atrioventricular (AV) counter state (AV_Count_STATE) is TRUE and last two guards state that the pacemaker's sensor and actuator $(PM_Sensor_A, PM_Actuator_A)$ of atrial are OFF. The actions of this event reset all the parameters of operating mode of the pacemaker system. The actions $(act1 - act5)$ of this event state that the pacemaker's actuator $(PM_Actuator_V)$ of ventricular sets OFF, assigns the value of variable (AV_count) as 0, atrioventricular (AV) counter state (AV_Count_STATE) sets FALSE, the pacemaker's sensor (PM_Sensor_V) of ventricular sets OFF and assigns the value of the current clock counter sp as 1.

EVENT Actuator_ON_A
WHEN
grd1 : $PM_Sensor_V = ON$
grd2 : $sp \geq Pace_Int - FixedAV \land sp \geq VRP \land sp \geq PVARP$
grd3 : $PM_Actuator_A = OFF$
grd4 : $PM_Sensor_A = ON$
THEN
act1 : $PM_Actuator_A := ON$
act2 : $PM_Sensor_V := OFF$
act3 : $PM_Sensor_A := OFF$
END

The actions $(act1{-}act3)$ of event $(Actuator_ON_A)$ state that the pacemaker's actuator $(PM_Actuator_A)$ of atrial sets ON and the pacemaker's sensor $(PM_Sensor_V, PM_Sensor_A)$ of ventricular and atrial set OFF when all guards satisfy. The first guard of this event states that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON, the next guard $(grd2)$ states that current clock counter sp is greater than or equal to ventriculoatrial (VA) interval, VRP and PVARP, the third guard shows that the pacemaker's actuator $(PM_Actuator_A)$ of atrial is OFF and and in last guard states that the pacemaker's sensor (PM_Sensor_A) of atrial is ON.

EVENT Actuator_OFF_A
WHEN
grd1 : $PM_Actuator_A = ON$
grd2 : $sp = Pace_Int - FixedAV \land sp \geq VRP \land sp \geq PVARP$
grd3 : $AV_Count_STATE = FALSE$
THEN
act1 : $PM_Actuator_A := OFF$
act2 : $AV_Count_STATE := TRUE$
END

The actions $(act1, act2)$ of event $(Actuator_OFF_A)$ state that the pacemaker's actuator $(PM_Actuator_A)$ of atria sets in OFF state and atrioventricular (AV) counter state (AV_Count_STATE) sets in TRUE state. The guards $(grd1, grd2)$ of this event state that the pacemaker's actuator $(PM_Actuator_A)$ of atria is ON, current clock counter sp is greater than or equal to ventriculoatrial (VA) interval, VRP and PVARP. In last guard states that atrioventricular (AV) counter states (AV_Count_STATE) is FALSE.

EVENT Sensor_ON_A
WHEN
grd1 : $PM_Sensor_A = OFF$
grd2 : $sp < Pace_Int - FixedAV \land sp \geq VRP \land sp \geq PVARP$
grd3 : $PM_Sensor_V = OFF$
THEN
act1 : $PM_Sensor_A := ON$
END

The events $(Sensor_ON_A$ and $Sensor_OFF_A)$ is used to control the sensing activities from the atrial chamber. The pacemaker's sensor (PM_Sensor_A) of atrial chamber synchronizes ON and OFF states under real time constraints. The guard $(grd1)$ of event $(Sensor_ON_A)$ represents that if the pacemaker's sensor (PM_Sensor_A) of atrial is OFF and next guard $(grd2)$ represents that the current clock counter sp is less than ventriculoatrial (VA) interval and greater than or equal to VRP and PVARP. The last guard $(grd3)$ represents that the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF. If all guards are true then in action part of this event the pacemaker's sensor (PM_Sensor_A) of atrial sets ON state.

EVENT Sensor_OFF_A
  WHEN
    grd1 : $PM_Sensor_A = ON$
    grd2 : $sp < Pace_Int - FixedAV \land sp \geq VRP \land sp \geq PVARP$
  THEN
    act1 : $PM_Sensor_A := OFF$
    act2 : $AV_Count_STATE := TRUE$
  END

The event $(Sensor_OFF_A)$ is used to set the pacemaker's sensor (PM_Sensor_A) of atrial in OFF state. The guards of this event represent that the pacemaker's sensor (PM_Sensor_A) of atrial is ON and current clock counter sp is less than ventriculoatrial (VA) interval and greater than or equal to VRP and PVARP. In actions of this event state that the pacemaker's sensor (PM_Sensor_A) of atrial sets OFF and atrioventricular (AV) counter state sets TRUE.

EVENT Sensor_ON_V
  WHEN
    grd1 : $PM_Sensor_V = OFF$
    grd2 : $(sp \geq VRP \land sp < Pace_Int - FixedAV \land PM_Sensor_A = ON)$
    $\lor$
    $(sp \geq Pace_Int - FixedAV \land AV_Count_STATE = TRUE)$
    grd3 : $PM_Actuator_A = OFF$
  THEN
    act1 : $PM_Sensor_V := ON$
  END

The events $(Sensor_ON_V$ and $Sensor_OFF_V)$ is used to control the sensing activities from the ventricular chamber. The pacemaker's sensor (PM_Sensor_V) of ventricular chamber synchronizes ON and OFF states under real time constraints. The guard $(grd1)$ of event $(Sensor_ON_V)$ represents that if the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF and in next guard $(grd2)$ shows that current clock counter sp is greater than or equal to VRP, less than ventriculoatrial (VA) interval and the pacemaker's sensor (PM_Sensor_A) of atrial is ON, or greater than or equal to ventriculoatrial (VA) interval and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE. The last guard $(grd3)$ states that the pacemaker's actuator $(PM_Actuator_A)$ of atrial is OFF. If all guards are true then in action part of this event, the pacemaker's sensor (PM_Sensor_V) of ventricular sets in ON state.

EVENT Sensor_OFF_V
  WHEN
    grd1 : $PM_Sensor_V = ON$
    grd2 : $sp \geq VRP \land sp \geq PVARP$
    grd3 : $(sp < Pace_Int - FixedAV)$
    $\lor$
    $(sp \geq Pace_Int - FixedAV \land sp < Pace_Int)$
    grd4: $PM_Actuator_V = OFF$
    grd5: $PM_Actuator_A = OFF$
  THEN
    act1 : $PM_Sensor_V := OFF$
    act2 : $AV_Count := 0$
    act3 : $AV_Count_STATE := FALSE$
    act4 : $last_sp := sp$
    act5 : $sp := 1$
    act6 : $PM_Sensor_A := OFF$
  END

The event $(Sensor_OFF_V)$ is used to set the pacemaker's sensor (PM_Sensor_V) of ventricular in OFF state. The guards $(grd1, grd2)$ of this event represent that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON and current clock counter sp is greater than or equal to VRP and PVARP. The next guard $(grd3)$ represents that the current clock counter sp is less than ventriculoatrial (VA) interval or greater than or equal to ventriculoatrial (VA) interval and less than pace interval $(Pace_Int)$. The last two guards $(grd4, grd5)$ state that the pacemaker's actuator $(PM_Actuator_V, PM_Actuator_A)$ of ventricular and atrial are OFF. The actions $(act1 - act6)$ of this event state that the pacemaker's sensor (PM_Sensor_V) of ventricular sets OFF, assigns the value of variable (AV_count) as 0, atrioventricular (AV) counter state (AV_Count_STATE) sets FALSE, assigns the value of the current clock counter sp to new variable $(last_sp)$, assigns the value of clock counter sp as 1 and sets OFF state of the pacemaker's actuator $(PM_Actuator_A)$ of the atrial chamber.

EVENT tic
WHEN
grd1 : $(sp < VRP)$
$\vee$
$(sp \geq VRP \wedge sp < Pace_Int - FixedAV \wedge PM_Sensor_V = ON$
THEN
act1 : $sp := sp + 1$
END

The event tic of this abstraction progressively increases the current clock counter sp under pre-defined pace interval $(Pace_Int)$. The guards of this event control the pacing stimulus into the heart chambers (atria and ventricular), synchronize ON and OFF states of the pacemaker's actuator $(PM_Actuator_A, PM_Actuator_V)$ of each chamber (atria and ventricular) and also control the sensing intrinsic stimulus from atrial and ventricular chambers and synchronize ON and OFF states of the pacemaker's sensor $(PM_Sensor_A,$ $PM_Sensor_V)$ in both chambers under real time constraints.

EVENT tic_AV
WHEN
grd1 : $AV_Count \leq FixedAV$
grd2 : $AV_Count_STATE = TRUE$
grd3 : $sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int$
THEN
act1 : $AV_Count := AV_Count + 1$
act2 : $sp := sp + 1$
END

The last event (tic_AV) of this abstraction progressively counts the atrioventricular (AV) interval and also increases the current clock counter sp is represented in actions ($act1$ and $act2$) . The guards of this event state that atrioventricular (AV) counter (AV_Count) is less than and equal to atrioventricular (AV) interval $(FixedAV)$, atrioventricular (AV) state (AV_count_STATE) is TRUE and the current clock counter sp is within the atrioventricular (AV) interval.

Abstraction of VDD mode:

In VDD operating mode of the double electrode pacemaker system, the first letter 'V' represents that the pacemaker paces ventricle only, second letter 'D' represents that the pacemaker senses both atrial and ventricle and final letter 'D' represents two conditional meaning that depends on atrial and ventricular sensing; first is that atrial sensing triggers ventricular pacing and second is that ventricular sensing inhibits ventricular pacing. In the block diagram (Fig-8) of heart pacing in VDD operating mode a new LRI follows the timing of each preceding LRI. When a sensed P wave occurs an atrioventricular (AV) interval is triggered within.

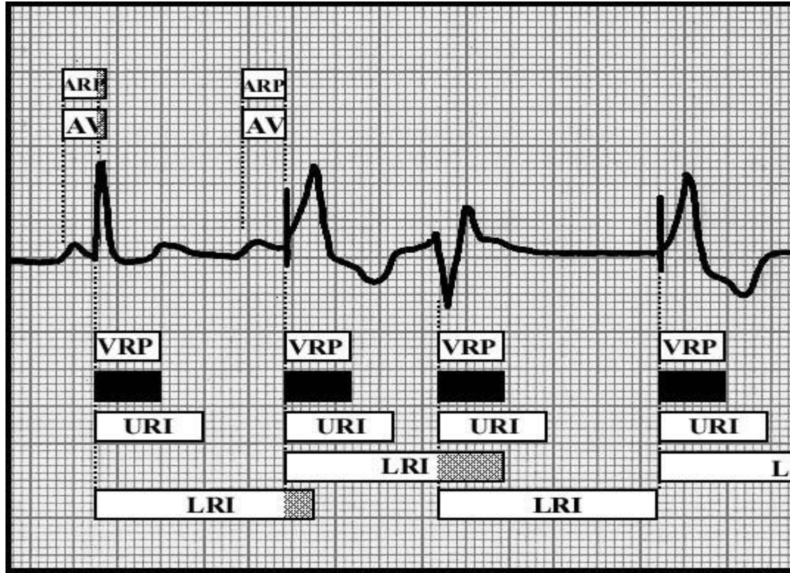


Fig. 8 Basic block diagram of ECG rhythm strip in VDD Operating Mode

In this abstract model, we have formalized a bradycardia operating mode *VDD* of the double electrode pacemaker. In this operating mode, the pacemaker uses actuators and sensors in both chambers. We have defined a new variable $PM_Actuator_V$ that represents ON or OFF states of pacemaker's actuators for pacing in the ventricular chamber. Next two variables PM_Sensor_A and PM_Sensor_V represent the ON or OFF states of pacemaker's sensor for sensing an intrinsic pulse from both the atrial and ventricular chambers. An interval between two paces is defined by a new variable $Pace_Int$ that must be between upper rate interval (URI) and lower rate interval (LRI), is represented by an invariant ($inv4$). A variable sp (since pace) represents the current *clock counter*. A variable $last_sp$ represents the last interval (in ms.) between two paces and a safety property in invariant ($inv6$) states that last interval must be between PVARP and pace interval $Pace_Int$. Another new variable AV_Count_STATE in invariant ($inv7$) is defined as boolean type to control the atrioventricular (AV) interval state and next variable AV_Count is defined as natural number to count the atrioventricular (AV) interval by invariant ($inv8$). Here new invariant $(inv10)$ represents the safety property and states that when clock counter sp is less then ventricular refractory period (VRP) and atrioventricular (AV) counter state (AV_Count_STATE) is $TRUE$, the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF and pacemaker's sensors of both chambers are OFF. The next invariant ($inv11$) represents that pacemaker's actuator ($PM_Actuator_V$) of ventricular is ON when clock counter sp is either equal to the pace interval $Pace_Int$ or clock counter sp less than pace interval $Pace_Int$ and atrioventricular (AV) counter AV_Count is greater then blanking period V_Blank and greater than or equal to the atrioventricular (AV) period $FixedAV$.

$inv1 : PM_Actuator_V \in status$
$inv2 : PM_Sensor_A \in status$
$inv3 : PM_Sensor_V \in status$
$inv4 : Pace_Int \in URI \,..\, LRI$
$inv5 : sp \in 1 \,..\, Pace_Int$
$inv6 : last_sp \geq PVARP \wedge last_sp \leq Pace_Int$
$inv7 : AV_Count_STATE \in BOOL$
$inv8 : AV_Count \in \mathbb{N}$
$inv9 : Pace_Int - FixedAV < Pace_Int$
$inv10 : sp < VRP \wedge AV_Count_STATE = FALSE$
$\Rightarrow$
$PM_Actuator_V = OFF \wedge$
$PM_Sensor_A = OFF \wedge$
$PM_Sensor_V = OFF$
$inv11 : PM_Actuator_V = ON$
$\Rightarrow$
$(sp = Pace_Int$
$\vee$
$(sp < Pace_Int \wedge$
$AV_Count > V_Blank \wedge$
$AV_Count \geq FixedAV))$

In the abstract specification of VDD operating mode, there are eight events $Actuator_ON_V$ to start pacing in ventricular, $Actuator_OFF_V$ to stop pacing in ventricular, $Sensor_ON_V$ to star sensing in ventricular, $Sensor_OFF_V$ to stop sensing in ventricular, $Sensor_ON_A$ to star sensing in atrial, $Sensor_OFF_A$ to stop sensing in atrial, tic to increment the current clock counter under real time constraints and tic_AV to count the atrioventricular (AV) interval.

EVENT Actuator_ON_V
WHEN
grd1 : $PM_Actuator_V = OFF$
grd2 : $(sp = Pace_Int)$
$\vee$
$(sp < Pace_Int \wedge AV_Count > V_Blank \wedge AV_Count \geq FixedAV)$
grd3 : $sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM_Actuator_V := ON$
act2 : $last_sp := sp$
END

The events $Actuator_ON_V$ and $Actuator_OFF_V$ start and stop the pacemaker's actuator $(PM_Actuator_V)$ in ventricular chamber and synchronizes ON and OFF states. The guard $(grd1)$ of event $Actuator_ON_V$ represents that when the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF and in guard $(grd2)$ the current clock counter sp is equal to pace interval $(Pace_Int)$ or less than pace interval $Pace_Int$ and atrioventricular (AV) counter (AV_Count) is greater than blanking period (V_Blank) and greater than or equal to atrioventricular (AV) interval $(FixedAV)$. In the last guard, the current clock counter sp is greater than or equal to ventricular refractory period (VRP) and post ventricular refractory period (PVARP). The actions of this event represent that if all guards are satisfy then the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and clock counter sp assigns to a new variable $(last_sp)$.

EVENT Actuator_OFF_V
WHEN
grd1 : $PM_Actuator_V = ON$
grd2 : $(sp = Pace_Int)$
$\vee$
$(sp \geq VRP \wedge sp < Pace_Int \wedge AV_Count > V_Blank \wedge$
$AV_Count \geq FixedAV \wedge AV_Count_STATE = TRUE)$
grd2 : $(sp \geq PVARP)$
THEN
act1 : $PM_Actuator_V := OFF$
act2 : $AV_Count := 0$
act3 : $AV_Count_STATE := FALSE$
act4 : $PM_Sensor_V := OFF$
act5 : $PM_Sensor_A := OFF$
act6 : $last_sp := sp$
act7 : $sp := 1$
END

The guards of event $Actuator_OFF_V$ states that the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and current clock counter sp is equal to the pace interval $Pace_Int$ or greater than or equal to VRP, less than pace interval $(Pace_Int)$, atrioventricular (AV) counter (AV_Count) is greater than blanking period (V_Blank), atrioventricular (AV) counter (AV_Count) is greater than or equal to the atrioventricular (AV)interval $(FixedAV)$ and atrioventricular (AV) counter state (AV_Count_State) is TRUE. The last guard states that current clock counter sp is greater than or equal to PVARP. The actions of this event reset the all parameters of the pacemaker for beginning the pacing cycle. In action $(act1)$ sets OFF state of the pacemaker's actuator $(PM_Actuator_V)$ of ventricular, in the second action reassign the value of variable (AV_count) as 0, in next action $(act3)$ sets FALSE state to the AV counter state AV_Count_STATE, sets OFF state to the pacemaker's sensor $(PM_Sensor_V, PM_Sensor_A)$ of both chambers in actions $(act4, act5)$, in action $(act6)$ assigns the value of the current clock counter sp to new variable $last_sp$ and finally in action $(act7)$ assigns the value of the current clock counter sp as 1.

EVENT Sensor_ON_A
WHEN
grd1 : $PM_Sensor_A = OFF$
grd2 : $sp < Pace_Int - FixedAV \wedge sp \geq VRP \wedge sp \geq PVARP$
grd3 : $PM_Sensor_V = OFF$
THEN
act1 : $PM_Sensor_A := ON$
END

The events $(Sensor_ON_A$ and $Sensor_OFF_A)$ is used to control the sensing activities from the atrial chamber. The pacemaker's sensor (PM_Sensor_A) of atrial chamber synchronizes ON and OFF states under real time constraints. The guard $(grd1)$ of event $(Sensor_ON_A)$ represents that if the pacemaker's sensor (PM_Sensor_A) of atrial is OFF and guard $(grd2)$ represents that the current clock counter sp is less than ventriculoatrial (VA) interval and greater than or equal to VRP and PVARP. The last guard $(grd3)$ represents that the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF. If all guards are true then in action part of this event then the pacemaker's sensor (PM_Sensor_A) of atrial sets in ON state.

EVENT Sensor_OFF_A
WHEN
grd1 : $PM_Sensor_A = ON$
grd2 : $sp < Pace_Int - FixedAV \wedge sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM_Sensor_A := OFF$
act2 : $AV_Count_STATE := TRUE$
END

The event $(Sensor_OFF_A)$ is used to set the pacemaker's sensor (PM_Sensor_A) of atrial in OFF state. The guards of this event show that the the pacemaker's sensor (PM_Sensor_A) of atrial is ON and the current clock counter sp is less than ventriculoatrial (VA) interval and greater than or equal to VRP and PVARP. In actions of this event state that the pacemaker's sensor (PM_Sensor_A) of atrial sets in OFF state and atrioventricular (AV) counter state (AV_Count_STATE) sets in TRUE state.

EVENT Sensor_ON_V
WHEN
grd1 : $PM_Sensor_V = OFF$
grd2 : $(sp \geq VRP \wedge sp < Pace_Int - FixedAV \wedge PM_Sensor_A = ON)$
$\vee$
$(sp \geq Pace_Int - FixedAV \wedge AV_Count_STATE = TRUE)$
THEN
act1 : $PM_Sensor_V := ON$
END

The events ($Sensor_ON_V$ and $Sensor_OFF_V$) is used to control the sensing activities from the ventricular chamber. The pacemaker's sensor (PM_Sensor_V) of ventricular chamber synchronizes ON and OFF states under real time constraints. The guard $(grd1)$ of event $(Sensor_ON_V)$ represents that if the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF and in next guard $(grd2)$ represents that the current clock counter sp is greater than or equal to VRP, less than ventriculoatrial (VA) interval and the pacemaker's sensors (PM_Sensor_A) of atrial is ON or current clock counter sp is greater than or equal to ventriculoatrial (VA) interval and atrioventricular (AV) counter state (AV_Count_STATE) is in TRUE state. If all guards are true then in action part of this event, the pacemaker's sensor (PM_Sensor_V) of ventricular sets in ON state.

EVENT Sensor_OFF_V
WHEN
grd1 : $PM_Sensor_V = ON$
grd2 : $sp \geq VRP \wedge sp \geq PVARP$
grd3 : $(sp < Pace_Int - FixedAV)$
$\vee$
$(sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int)$
grd4: $PM_Actuator_V = OFF$
THEN
act1 : $PM_Sensor_V := OFF$
act2 : $AV_Count := 0$
act3 : $AV_Count_STATE := FALSE$
act4 : $last_sp := sp$
act5 : $sp := 1$
act6 : $PM_Sensor_A := OFF$
END

The event $(Sensor_OFF_V)$ is used to set the pacemaker's sensor (PM_Sensor_V) of ventricular in OFF state. The guard $(grd1)$ of this event represents that the pacemaker's sensor (PM_Sensor_V) of

ventricular is ON and the current clock counter sp is greater than or equal to VRP and PVARP interval. The next guard $(grd3)$ represents that the current clock counter sp is less than ventriculoatrial (VA) interval or greater than or equal to ventriculoatrial (VA) interval and less than automatic pace interval $(Pace_Int)$. The last guard $(grd4)$ state that the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF. All actions of this event is same as event $(Sensor_OFF_V)$ of DDI operating mode which are already described in actions part of this event $(Sensor_OFF_V)$.

EVENT tic
WHEN
grd1 : $(sp < VRP)$
$\vee$
$(sp \geq VRP \wedge sp < Pace_Int \wedge PM_Sensor_V = ON \wedge$
$PM_Sensor_A = ON$
THEN
act1 : $sp := sp + 1$
END

The event (tic) of this abstraction progressively increases the current clock counter sp under pre-defined pace interval $(Pace_Int)$. The guard of this event control the pacing stimulus into the heart chambers (atria and ventricular), synchronizes ON and OFF states of the pacemaker's actuator $(PM_Actuator_V)$ of ventricular chamber and also control the sensing intrinsic stimulus of the atrial and ventricular chamber and synchronize ON and OFF states of the pacemaker's sensor $(PM_Sensor_A, PM_Sensor_V)$ in both chambers under real time constraints.

EVENT tic_AV
WHEN
grd1 : $AV_Count < FixedAV$
grd2 : $AV_Count_STATE = TRUE$
grd3 : $(sp \geq VRP \wedge sp \geq PVARP \wedge sp < Pace_Int - FixedAV)$
$\vee$
$(sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int)$
THEN
act1 : $AV_Count := AV_Count + 1$
act2 : $sp := sp + 1$
END

The last event (tic_AV) of this abstraction progressively counts the atrioventricular (AV) interval and also increases the current clock counter sp is represented in actions $(act1$ and $act2)$. The guards of this event states that atrioventricular (AV) counter (AV_Count) is less than atrioventricular (AV) interval $(FixedAV)$, atrioventricular (AV) state (AV_count_STATE) is in TRUE state and the current clock counter sp is within the ventriculoatrial (VA) interval $(Pace_Int - FixedAV)$ and greater than or equal to VRP and PVARP interval or clock counter sp is greater than or equal to atrioventricular (AV) interval and less than pace interval $(Pace_Int)$.

Abstraction of DDD mode:

In DDD operating mode of double electrode pacemaker system, the first letter 'D' represents that the pacemaker paces in both atrial and ventricle chambers, second letter 'D' represents that the pacemaker senses intrinsic activities from both atrial and ventricle chambers and final letter 'D' represents two conditional meaning that depends on atrial and ventricular sensing; first is that atrial sensing inhibits atrial pacing and triggers ventricular pacing and second is that ventricular sensing inhibits ventricular and atrial pacing. In the block diagram (Fig-9) of heart pacing in DDD operating mode a ventriculoatrial (VA) interval follows the timing for each atrioventricular (AV) interval and an atrioventricular (AV) interval follows the timing for each ventriculoatrial (VA) interval, except with a 'P' wave or an 'R' wave (PVC) that starts timing of a new ventriculoatrial (VA) interval.

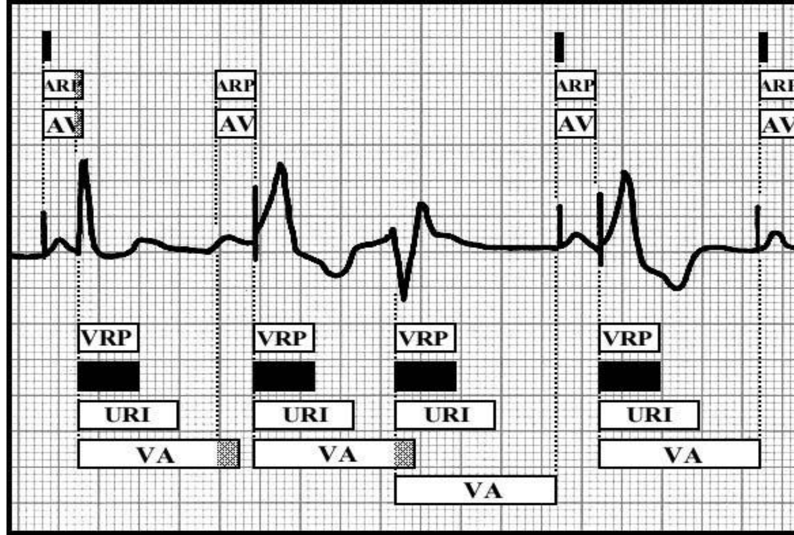


Fig. 9 Basic block diagram of ECG rhythm strip in DDD Operating Mode

In this abstract model, we have formalized a bradycardia operating mode *DDD* of the double electrode pacemaker. In this operating mode, the pacemaker uses actuators and sensors in both chambers. We have defined two new variables $PM_Actuator_A$ and $PM_Actuator_V$ that represent ON or OFF states of pacemaker's actuators for pacing in the atrial and ventricular chambers. Similarly next two variables PM_Sensor_A and PM_Sensor_V represent ON or OFF states of pacemaker's sensor for sensing an intrinsic pulse from both the atrial and ventricular chambers. An interval between two paces is defined by a new variable $Pace_Int$ that must be between upper rate interval (URI) and lower rate interval (LRI), is represented by an invariant ($inv5$). A variable sp (since pace) represents the current *clock counter*. A variable $last_sp$ represents the last interval (in ms.) between two paces and a safety property in invariant ($inv7$) states that last interval must be between PVARP and pace interval $Pace_Int$. Another new variable AV_Count_STATE is defined as boolean type to control the atrioventricular (AV) interval state and next variable AV_Count is defined as natural number to count the atrioventricular (AV) interval. The invariants ($inv11$,$inv12$ and $inv13$) represent the safety properties. The invariant $inv11$ states that when clock counter sp is less than ventricular refractory period (VRP) and atrioventricular (AV) counter state AV_Count_State is $FALSE$, pacemaker's actuators and sensors of both chambers are OFF. Similarly,the next invariants ($inv12$ and $inv13$) represent the conditions of ON state of the pacemaker's actuators in both chambers

$$
\begin{array}{l}
inv1 : PM_Actuator_A \in status \\
inv2 : PM_Actuator_V \in status \\
inv3 : PM_Sensor_A \in status \\
inv4 : PM_Sensor_V \in status \\
inv5 : Pace_Int \in URI \mathrel{..} LRI \\
inv6 : sp \in 1 \mathrel{..} Pace_Int \\
inv7 : last_sp \geq PVARP \wedge last_sp \leq Pace_Int \\
inv8 : AV_Count_STATE \in BOOL \\
inv9 : AV_Count \in \mathbb{N} \\
inv10 : Pace_Int - FixedAV < Pace_Int \\
inv11 : sp < VRP \wedge AV_Count_STATE = FALSE \Rightarrow \\
\qquad PM_Actuator_V = OFF \wedge PM_Sensor_A = OFF \wedge \\
\qquad PM_Sensor_V = OFF \wedge PM_Actuator_A = OFF \\
inv12 : PM_Actuator_V = ON \Rightarrow \\
\qquad sp = Pace_Int \vee (sp < Pace_Int \wedge \\
\qquad AV_Count > V_Blank \wedge AV_Count \geq FixedAV) \\
inv13 : PM_Actuator_A = ON \Rightarrow \\
\qquad (sp \geq Pace_Int - FixedAV)
\end{array}
$$

In the abstract specification of DDD operating mode, there are ten events $Actuator_ON_A$ to start pacing in atrial, $Actuator_OFF_A$ to stop pacing in atrial, $Actuator_ON_V$ to start pacing in ventricular, $Actuator_OFF_V$ to stop pacing in ventricular, $Sensor_ON_V$ to start sensing in ventricular, $Sensor_OFF_V$ to stop sensing in ventricular, $Sensor_ON_A$ to star sensing in atrial, $Sensor_OFF_A$ to stop sensing in atrial, tic to increment the current clock counter sp under real time constraints and tic_AV to count the atrioventricular (AV) interval.

EVENT Actuator_ON_V
WHEN
grd1 : $PM_Actuator_V = OFF$
grd2 : $(sp = Pace_Int)$
$\vee$
$(sp < Pace_Int \wedge AV_Count > V_Blank \wedge$
$AV_Count \geq FixedAV)$
grd3 : $sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM_Actuator_V := ON$
act2 : $last_sp := sp$
END

The events $Actuator_ON_V$ and $Actuator_OFF_V$ start and stop the pacemaker's actuator in ventricular chamber and synchronize ON and OFF states. The guard $(grd1)$ of event $Actuator_ON_V$ represents that the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is OFF. In guard (grd2), the current clock counter sp is equal to pace interval $(Pace_Int)$ or less than pace interval $(Pace_Int)$ and atrioventricular (AV) counter (AV_Count) is greater than blanking period (V_Blank) and greater than or equal to atrioventricular (AV) interval $(FixedAV)$. In last guard $(grd3)$, the current clock counter sp is greater than or equal to ventricular refractory period (VRP) and post ventricular refractory period (PVARP). The actions represent that if all guards are satisfy then the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and the current clock counter sp assigns to new variable $(last_sp)$.

EVENT Actuator_OFF_V
WHEN
grd1 : $PM_Actuator_V = ON$
grd2 : $(sp = Pace_Int)$
$\vee$
$(sp < Pace_Int \wedge AV_Count > V_Blank \wedge$
$AV_Count \geq FixedAV$
grd3 : $AV_Count_STATE = TRUE$
grd4 : $PM_Actuator_A = OFF$
grd5 : $PM_Sensor_A = OFF$
THEN
act1 : $PM_Actuator_V := OFF$
act2 : $AV_Count := 0$
act3 : $AV_Count_STATE := FALSE$
act4 : $PM_Sensor_V := OFF$
act5 : $sp := 1$
END

The guards of event $(Actuator_OFF_V)$ states that the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON and current clock counter sp is equal to pace interval $(Pace_Int)$ or less than pace interval $(Pace_Int)$, atrioventricular (AV) counter (AV_Count) is greater than blanking period (V_Blank) and atrioventricular (AV) counter is greater than or equal to atrioventricular (AV)interval $(FixedAV)$. The guard $(grd3)$ states that atrioventricular (AV) counter state (AV_Count_STATE) is TRUE and last two guards represent that the pacemaker's actuator and sensor $(PM_Actuator_A, PM_Sensor_A)$ of atrial chamber are OFF. The actions of this event reset the all parameters of the pacemaker. In actions part, sets

OFF state of the pacemaker's actuator ($PM_Actuator_V$) of ventricular, reassigns the value of variable (AV_count) as 0, sets FALSE state to the AV counter state (AV_Count_STATE), sets an OFF state to the pacemaker's sensor (PM_Sensor_V) of ventricular chamber and finally assigns the value of the current clock counter sp as 1.

EVENT Actuator_ON_A
WHEN
grd1 : $PM_Sensor_V = ON$
grd2 : $sp \geq Pace_Int - FixedAV \land$
$sp \geq VRP \land sp \geq PVARP$
grd3 : $PM_Actuator_A = OFF$
grd4 : $PM_Sensor_A = ON$
THEN
act1 : $PM_Actuator_A := ON$
act2 : $PM_Sensor_V := OFF$
act3 : $PM_Sensor_A := OFF$
END

The actions ($act1-act3$) of event ($Actuator_ON_A$) state that the pacemaker's actuator ($PM_Actuator_A$) of atria is ON and pacemaker's sensors (PM_Sensor_V, PM_Sensor_A) of ventricular and atrial is OFF when all guards are satisfied. The first guard states that pacemaker's sensor (PM_Sensor_V) of ventricular is ON, the next guard ($grd2$) states that current clock counter sp is greater than or equal to ventriculoatrial (VA) interval, VRP and PVARP. The last two guards show that the pacemaker's actuator ($PM_Actuator_A$) of atrial is OFF and pacemaker's sensor (PM_Sensor_A) of atrial is ON.

EVENT Actuator_OFF_A
WHEN
grd1 : $PM_Actuator_A = ON$
grd2 : $AV_Count_STATE = FALSE$
grd3 : $sp \geq Pace_Int - FixedAV \land$
$sp \geq VRP \land sp \geq PVARP$
THEN
act1 : $PM_Actuator_A := OFF$
act2 : $AV_Count_STATE := TRUE$
END

In event $Actuator_OFF_A$, the actions ($act1, act2$) state that pacemaker's actuator ($PM_Actuator_A$) of atria is OFF and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE when the guards are satisfied. The first two guards ($grd1, grd2$) state that pacemaker's actuator ($PM_Actuator_A$) of atrial is ON and atrioventricular (AV) counter state (AV_Count_STATE) is $FALSE$. The last guard represents clock counter sp is greater than or equal to ventriculoatrial (VA) interval, VRP and PVARP.

EVENT Sensor_ON_V
WHEN
grd1 : $PM_Sensor_V = OFF$
grd2 : $(sp \geq VRP \land sp < Pace_Int - FixedAV \land PM_Sensor_A = ON)$
$\lor$
$(sp \geq Pace_Int - FixedAV \land$
$AV_Count_STATE = TRUE)$
grd3 : $PM_Actuator_A = OFF$
THEN
act1 : $PM_Sensor_V := ON$
END

The events ($Sensor_ON_V$ and $Sensor_OFF_V$) is used to control the sensing activities from the ventricular chamber. The pacemaker's sensor (PM_Sensor_V) of ventricular chamber synchronizes ON and OFF states under real time constraints. The guard ($grd1$) of event ($Sensor_ON_V$) represents that if the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF and next guard ($grd2$) represents that the current clock counter sp is greater than or equal to VRP, less than ventriculoatrial (VA) interval and the pacemaker's sensor (PM_Sensor_A) of atrial is ON or greater than or equal to ventriculoatrial (VA) interval and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE. The last guard ($grd3$) states that the pacemaker's actuator ($PM_Actuator_A$) of atrial is OFF. If all guards are true then in action part of this event, the pacemaker's sensor (PM_Sensor_V) of ventricular sets in ON state.

EVENT Sensor_OFF_V
WHEN
grd1 : $PM_Sensor_V = ON$
grd2 : $sp \geq VRP \wedge sp \geq PVARP$
grd3 : $(sp < Pace_Int - FixedAV)$
$\vee$
$(sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int)$
grd4: $PM_Actuator_V = OFF$
grd5: $PM_Actuator_A = OFF$
THEN
act1 : $PM_Sensor_V := OFF$
act2 : $AV_Count := 0$
act3 : $AV_Count_STATE := FALSE$
act4 : $last_sp := sp$
act5 : $sp := 1$
act6 : $PM_Sensor_A := OFF$
END

The event ($Sensor_OFF_V$) is used to set the pacemaker's sensor (PM_Sensor_V) of ventricular in OFF state. The guards ($grd1, grd2$) of this event represent that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON and the current clock counter sp is greater than or equal to VRP and PVARP. The next guard ($grd3$) represents that the current clock counter sp is less than ventriculoatrial (VA) interval or greater than or equal to ventriculoatrial (VA) interval and less than pace interval ($Pace_Int$). The last two guards ($grd4, grd5$) state that the pacemaker's actuator ($PM_Actuator_V, PM_Actuator_A$) of ventricular and atrial are OFF. The actions ($act1-act6$) of this event state that the pacemaker's sensor (PM_Sensor_V) of ventricular sets in OFF state, assigns the value of variable (AV_count) as 0, atrioventricular (AV) counter state (AV_Count_STATE) sets in FALSE state, assigns the value of clock counter sp to new variable ($last_sp$), assigns the value of the current clock counter sp as 1 and sets OFF state of the pacemaker's actuator ($PM_Actuator_A$) of atrial.

EVENT Sensor_ON_A
WHEN
grd1 : $PM_Sensor_A = OFF$
grd2 : $PM_Sensor_V = OFF$
grd3 : $sp < Pace_Int - FixedAV \wedge$
$sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM_Sensor_A := ON$
END

The events ($Sensor_ON_A$ and $Sensor_OFF_A$) is used to control the sensing activities from the atrial chamber. The pacemaker's sensor (PM_Sensor_A) of atrial chamber synchronizes ON and OFF states under real time constraints. The guard ($grd1$) of event ($Sensor_ON_A$) represents that if the pacemaker's sensor (PM_Sensor_A) of atrial is OFF and second guard ($grd2$) represents that the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF.In last guard ($grd3$) represents that the current clock counter sp is

less than ventriculoatrial (VA) interval and greater than or equal to VRP and PVARP interval. If all guards are true then in action part of this event, pacemaker's sensor (PM_Sensor_A) of atrial sets in ON state.

> **EVENT Sensor_OFF_A**
> **WHEN**
> grd1 : $PM_Sensor_A = ON$
> grd2 : $sp < Pace_Int - FixedAV \land$
> $sp \geq VRP \land sp \geq PVARP$
> **THEN**
> act1 : $PM_Sensor_A := OFF$
> act2 : $AV_Count_STATE := TRUE$
> **END**

The event ($Sensor_OFF_A$) is used to set the pacemaker's sensor (PM_Sensor_A) of atrial in OFF state. The guards of this event represent that the pacemaker's sensor (PM_Sensor_A) of atrial is ON and the current clock counter sp is less than ventriculoatrial (VA) interval and greater than or equal to VRP and PVARP. In actions of this event state that the pacemaker's sensor (PM_Sensor_A) of atrial sets in OFF state and atrioventricular (AV) counter state (AV_Count_STATE) sets TRUE.

> **EVENT tic**
> **WHEN**
> grd1 : $(sp < VRP)$
> $\lor$
> $(sp \geq VRP \land sp < Pace_Int - FixedAV \land$
> $PM_Sensor_A = ON \land PM_Sensor_V = ON$
> **THEN**
> act1 : $sp := sp + 1$
> **END**

The event (tic) of this abstraction progressively increases the current clock counter sp under pre-defined pace interval ($Pace_Int$). The guards of this event control the pacing stimulus into the heart chambers (atria and ventricular), synchronizes ON and OFF states of the pacemaker's actuator ($PM_Actuator_A, PM_Actuator_V$) of each chamber (atria and ventricular) and also control the sensing intrinsic stimulus of atrial and ventricular chamber and synchronizes ON and OFF states of the pacemaker's sensor (PM_Sensor_A, PM_Sensor_V) in atrial and ventricular under real time constraints.

> **EVENT tic_AV**
> **WHEN**
> grd1 : $AV_Count < FixedAV$
> grd2 : $AV_Count_STATE = TRUE$
> grd3 : $(sp \geq VRP \land sp \geq PVARP \land$
> $sp < Pace_Int - FixedAV)$
> $\lor$
> $(sp \geq Pace_Int - FixedAV \land$
> $sp < Pace_Int)$
> **THEN**
> act1 : $AV_Count := AV_Count + 1$
> act2 : $sp := sp + 1$
> **END**

The last event (tic_AV) of this abstraction progressively counts the atrioventricular (AV) interval and also increases the current clock counter (sp) is represented in actions ($act1$ and $act2$) . The guards of this event states that atrioventricular (AV) counter (AV_Count) is less than and equal to atrioventricular (AV) interval ($FixedAV$), atrioventricular (AV) state (AV_count_STATE) is TRUE and the current clock counter sp

is within the ventriculoatrial (VA) interval $(Pace_Int - FixedAV)$ and greater than or equal to VRP and PVARP or the current clock counter sp is greater than or equal to atrioventricular (AV) interval and less than pace interval $(Pace_Int)$.

5.4.2 First refinement:Threshold

The pacemaker control unit delivers stimulation to the heart chambers, on the basis of measured threshold value under safety margin. We define two new constants STA_THR_A and STA_THR_V to hold the standard threshold value in axioms ($axm1$ and $axm2$). The threashold constants are different for the atria and the ventricular chambers.

$$
\begin{array}{l}
axm1 : STA_THR_A \in nat_1 \land STA_THR_A = 75 \\
axm1 : STA_THR_V \in nat_1 \land STA_THR_V = 250
\end{array}
$$

The pacemaker's sensor starts sensing after the refractory period; Atria Refractory Period (ARP) , Ventricular Refractory Period (VRP). The pacemaker's actuator delivers a pacing stimulus when sensing value is greater than or equal to the standard threshold constants STA_THR_A or STA_THR_V. In DOO operating mode only pacemaker's actuators paces in atrial and ventricular chambers under automatic pace interval without using any pacemaker's sensors, so in this mode no any refinement related to the threashold.

First refinement of DVI mode:

In the first refinement of DVI operating mode, we formalize the concept of sensing threshold value in the double electrode pacemaker. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of heart and a pulse generator for deliverying stimulation pulses to the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value under safety margin. We introduce the new variable (Thr_V) to hold the sensing threashold value of the pacemaker's sensor of ventricular chamber and next variable (Thr_V_State) represents the TRUE and FALSE states of the pacemaker's sensor to sense the intrinsic activity of the ventricular chamber.

$$
\begin{array}{l}
inv1 : Thr_V \in \mathbb{N}_1 \\
inv2 : Thr_V_State \in BOOL \\
inv3 : sp > VRP \land sp < Pace_Int - FixedAV \Rightarrow PM_Sensor_V = ON \\
inv4 : PM_Actuator_V = ON \Rightarrow sp = Pace_Int \\
inv5 : sp > VRP \land sp < Pace_Int \land Thr_V \geq STA_THR_V \land \\
\qquad Thr_V_State = TRUE \Rightarrow PM_Sensor_V = OFF \\
inv6 : sp > Pace_Int - FixedAV \land sp < Pace_Int \land \\
\qquad AV_Count_STATE = TRUE \Rightarrow PM_Actuator_V = OFF \\
inv7 : sp > Pace_Int - FixedAV \land sp < Pace_Int \land \\
\qquad AV_Count_STATE = TRUE \Rightarrow PM_Actuator_A = OFF \\
inv8 : sp > Pace_Int - FixedAV \land sp < Pace_Int \land \\
\qquad AV_Count_STATE = TRUE \Rightarrow PM_Sensor_V = ON \\
inv9 : PM_Actuator_A = ON \Rightarrow sp \geq Pace_Int âFixedAV â§spâVRPâ§sp < Pace_Int
\end{array}
$$

From invariants $(inv3 - inv9)$ represent the safety properties of the pacemaker system under pacing and sensing activities of electrode in DVI operating mode. The third invariant $(inv3)$ states that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON when current clock counter (sp) is greater than VRP and less than ventriculoatrial(VA) interval. The fourth invariant state that the pacemaker's actuator $(PM_Actuator_V)$ of ventricular is ON when current clock counter sp is equal to the pace interval $Pace_Int$. The invariant $(inv5)$ represents that the pacemaker's sensor (PM_Sensor_V) of ventricular is OFF when clock counter sp is greater then VRP, less then pace interval $(Pace_Int)$, sensed value (Thr_V) is greater than or equal to standard threshold (STA_THR_V) value of ventricular chamber. The next three invariants $(inv6, inv7$ and $inv8)$ represent that the pacemaker's actuator $(PM_Actuator_A, PM_Actuator_V)$ of atrial and ventricular are OFF and the pacemaker's sensor (PM_Sensor_V) of ventricular is ON when

clock counter sp is greater than ventriculoatrial (VA) interval, less than pace interval $(Pace_Int)$ and atrioventricular (AV) counter state (AV_Count_STATE) is $TRUE$. The last invariant $(inv9)$ states that the pacemaker's actuator of atrial chambers can be ON when current clock counter sp is within the ventriculoatrial (VA) interval $(Pace_Int - FixedAV)$ and greater than or equal to VRP and less than pace interval $Pace_Int$.

In this refinement we introduce the new event (Thr_Value_V) for sensing the intrinsic activities of ventricular chamber. This event is synchronized with all other events of this operating mode under all safety properties and real time constraints. The guards $(grd2 - grd4)$ of this event state that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON, threashold state (Thr_V_State) of ventricular is TRUE and sensed value (Thr_V) is less than standard threshold value (STA_THR_V) of ventricular chamber. The last guard states that either the current clock counter sp is greater than or equal to VRP and less than ventriculoatrial (VA) interval or the current clock counter sp is greater then and equal to atrioventricular (AV) interval and less then pace interval $(Pace_Int)$. The actions $(act1 - act2)$ of this event state that actual sensed value (Thr_V_val) of ventricular chamber assigns to variable (Thr_V) and sets the FALSE state of threashold ventricular state (Thr_V_State).

EVENT Thr_Value_V
WHEN
grd1 : $Thr_V_val \in \mathbb{N}$
grd2 : $PM_Sensor_V = ON$
grd3 : $Thr_V_State = TRUE$
grd4 : $Thr_V < STA_THR_V$
grd5 : $(sp \geq VRP \wedge sp < Pace_Int - FixedAV)$
$\vee$
$(sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int)$
THEN
act1 : $Thr_V := Thr_V_val$
act2 : $Thr_V_State := FALSE$
END

We add some new actions in events $(Actuator_OFF_V$, $Sensor_ON_V$, and $Sensor_OFF_V)$[1] to synchronize the sensing activities using event (Thr_V_val) under real time constraints which are already defined in the abstract model of this operating mode.

EVENT Actuator_OFF_V
$\oplus$ act9 : $Thr_V := 0$
$\oplus$ act10 : $Thr_V_State := FALSE$

EVENT Sensor_ON_V
$\oplus$ act2 : $Thr_V_State := FALSE$

EVENT Sensor_OFF_V
$\oplus$ act8 : $Thr_V := 0$

The event (tic) of this refinement model progressively increases the current clock counter sp under predefined pace interval $(Pace_Int)$. The guard $(grd1)$ of this event controls the pacing stimulus into the heart chambers (atria and ventricular), synchronizes ON and OFF states of pacemaker's actuators $(PM_Actuator_A, PM_Actuato$ of each chamber (atria and ventricular) and also control the sensing intrinsic stimulus of ventricular chamber and synchronizes ON and OFF states of the pacemaker's sensor (PM_Sensor_V) in ventricular under real time constraints. We modify the guard $(grd1)$ of this event and add more properties to synchronize the pacing and sensing activities and we also add new action $(act2)$. The additional guards and action handle the behavior of event (Thr_V_val) to sense the intrinsic activities from the ventricular chamber at actual required time.

[1]$\oplus$: To add a new guard and an action.

EVENT tic
  WHEN
    grd1 : $(sp < VRP \wedge PM_Sensor_V = OFF \wedge$
    $AV_Count_STATE = FALSE)$
    $\vee$
    $(sp \geq VRP \wedge sp < Pace_Int - FixedAV \wedge$
    $PM_Sensor_V = ON \wedge AV_Count_STATE = FALSE \wedge$
    $Thr_V_State = FALSE \wedge Thr_V < STA_THR_V))$
  THEN
$\oplus$ act2 : $Thr_V_State := TRUE$
  END

We add some new guards $(grd4 - grd8)$ and an action $(act3)$ in event (tic_AV) of this refinement. The new guards provide more specific and stronger guards to count the atrioventricular (AV) interval and action $(act3)$ states that threashold state (Thr_V_State) of ventricular is TRUE when all guards are satisfied.

EVENT tic_AV
  WHEN
$\oplus$ grd4 $PM_Sensor_V = ON$
$\oplus$ grd5 $Thr_V_State = FALSE$
$\oplus$ grd6 $Thr_V < STA_THR_V$
$\oplus$ grd7 $PM_Actuator_V = OFF$
$\oplus$ grd8 $PM_Actuator_A = OFF$
  THEN
$\oplus$ act3 : $Thr_V_State := TRUE$
  END

First refinement of DDI mode:

In the first refinement of DDI operating mode, we formalize the concept of sensing threshold value of the double electrode pacemaker. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of heart and a pulse generator for deliverying stimulation pulses to the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value under safety margin. We introduce the new variables $(Thr_A$ and $Thr_V)$ to hold the sensing threashold value of the pacemaker's sensor $(PM_Sensor_A, PM_Sensor_V)$ of atrial and ventricular chambers. Similarly next variables $(Thr_A_State$ and $Thr_V_State)$ represent TRUE or FALSE states of the pacemaker's sensor $(PM_Sensor_A, PM_Sensor_V)$ to sense the intrinsic activity of the atrial and ventricular chambers.

$inv1 : Thr_A \in \mathbb{N}_1$
$inv2 : Thr_V \in \mathbb{N}_1$
$inv3 : Thr_A_State \in BOOL$
$inv4 : Thr_V_State \in BOOL$
$inv5 : PM_Actuator_V = ON \Rightarrow sp = Pace_Int$
$inv6 : sp > VRP \wedge sp < Pace_Int - FixedAV \Rightarrow PM_Actuator_A = OFF$
$inv7 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge$
  $AV_Count_STATE = TRUE \Rightarrow PM_Sensor_V = ON$
$inv8 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge$
  $AV_Count_STATE = TRUE \Rightarrow PM_Actuator_V = OFF$
$inv9 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge$
  $AV_Count_STATE = TRUE \Rightarrow PM_Actuator_A = OFF$
$inv10 : PM_Actuator_A = ON \Rightarrow sp = Pace_Int - FixedAV$

From invariants ($inv5 - inv10$) represent the safety properties of the pacemaker system under pacing and sensing activities of electrode in DDI operating mode. The fifth invariant ($inv5$) state that the pacemaker's actuator ($PM_Actuator_V$) of ventricular is ON when current clock counter sp is equal to the pace interval $Pace_Int$. The next invariant ($inv6$) states that the pacemaker's actuator ($PM_Actuator_A$) of atrial is ON when current clock counter sp is greater than VRP and less than ventriculoatrial(VA) interval. The next three invariants ($inv7, inv8, inv9$) represent that the pacemaker's actuator ($PM_Actuator_A, PM_Actuator_V$) of atrial and ventricular chambers are OFF and the pacemaker's sensor (PM_Sensor_V) of ventricular is ON when current clock counter (sp) is greater than ventriculoatrial (VA) interval, less than pace interval ($Pace_Int$) and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE. The last invariant states that the pacemaker's actuator of atrial chamber is ON when current clock counter sp is equal to ventriculoatrial (VA) interval ($Pace_Int - FixedAV$).

EVENT Thr_Value_V
WHEN
grd1 : $Thr_V_val \in \mathbb{N}$
grd2 : $PM_Sensor_V = ON$
grd3 : $Thr_V_State = TRUE$
grd4 : $Thr_V < STA_THR_V$
grd5 : $(sp \geq VRP \wedge sp < Pace_Int - FixedAV)$
$\vee$
$(sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int)$
grd6 $(Thr_A_State = FALSE \wedge Thr_A < STA_THR_A)$
$\vee$
$PM_Sensor_A = OFF \wedge AV_Count < FixedAV$
THEN
act1 : $Thr_V := Thr_V_val$
act2 : $Thr_V_State := FALSE$
END

In this refinement we introduce the two new events (Thr_Value_V and Thr_Value_A) for sensing the intrinsic activities from ventricular and atrial chambers. These events are synchronized with all other events of this operating mode under all safety properties and real time constraints. The guards ($grd2 - grd4$) of event (Thr_V_val) state that pacemaker's sensor (PM_Sensor_V) of ventricular is ON, threashold state (Thr_V_State) of ventricular is TRUE and sensed value (Thr_V) is less than standard threshold value (STA_THR_V) of ventricular chamber. The next guard ($grd5$) represents that either clock counter (sp) is greater than or equal to VRP and less than ventriculoatrial (VA) interval or clock counter (sp) is greater than or equal to atrioventricular (AV) interval and less then pace interval ($Pace_Int$). The last guard ($grd6$) states that either threshold state (Thr_A_State) of atrial chamber is FLASE and threashold value (Thr_A) of atrial is less then standard threashold value (STA_THR_A) of atrial chamber or the pacemaker's sensor (PM_Sensor_A) of atrial chamber is OFF and atrioventricular (AV) counter (AV_Count) is less than atrioventricular (AV) interval ($FixedAV$). The actions ($act1 - act2$) of this event state that actual sensed value (Thr_V_val) of ventricular chamber assigns to variable Thr_V and sets FALSE state of threashold ventricular state (Thr_V_State).

EVENT Thr_Value_A
WHEN
grd1 : $Thr_A_val \in \mathbb{N}$
grd2 : $PM_Sensor_A = ON$
grd3 : $Thr_A_State = TRUE$
grd4 : $Thr_A < STA_THR_A$
grd5 : $(sp \geq VRP \wedge sp < Pace_Int - FixedAV)$
THEN
act1 : $Thr_A := Thr_A_val$
act2 : $Thr_A_State := FALSE$
END

Other new event (Thr_Value_A) introduce to take the intrinsic activities of the atrial chamber. The guards ($grd2 - grd4$) state that the pacemaker's sensor (PM_Sensor_A) of the atrial chamber is ON, threashold state of atrial chamber is TRUE and sensed value (Thr_A) of atrial chamber is less than standard threashold (STA_THR_A) of atrial chamber. The last guard ($grd5$) of this event state that the current clock counter sp is greater than or equal to VRP and less than ventriculoatrial (VA) interval. The actions ($act1 - act2$) of this event state that actual sensed value (Thr_A_val) of atrial chamber assigns to variable (Thr_A) and sets FALSE state of threashold ventricular state (Thr_A_State).

EVENT Actuator_OFF_V
⊕ act6 : $Thr_A := 0$
⊕ act7 : $Thr_V := 0$
⊕ act8 : $Thr_A_State := FALSE$
⊕ act9 : $Thr_V_State := FALSE$

EVENT Sensor_ON_A
⊕ act2 : $Thr_A_State := TRUE$

EVENT Sensor_OFF_V
⊕ grd6 : $Thr_V \geq STA_THR_V$
⊕ act7 : $Thr_A := 0$
⊕ act8 : $Thr_V := 0$
⊕ act9 : $Thr_A_State := FALSE$
⊕ act10 : $Thr_V_State := FALSE$

We add some new actions and guards in events ($Actuator_OFF_V$, $Sensor_ON_A$, and $Sensor_OFF_V$) to synchronize the sensing activities using events (Thr_A_val and Thr_V_val) under real time constraints, which are already defined in the abstract model of this operating mode.

EVENT tic
WHEN
grd1 : $(sp < VRP \wedge AV_Count_STATE = FALSE)$
$\vee$
$(sp \geq VRP \wedge sp < Pace_Int - FixedAV \wedge$
$PM_Sensor_V = ON \wedge PM_Actuator_A = OFF \wedge$
$Thr_V_State = FALSE \wedge Thr_V < STA_THR_V))$
THEN
⊕ act2 : $Thr_A_State := TRUE$
⊕ act3 : $Thr_V_State := TRUE$
END

The event (tic) of this refinement model progressively increases the current clock counter sp under predefined pace interval ($Pace_Int$). The guard of this event controls the pacing stimulus into the heart chambers (atria and ventricular), synchronizes ON and OFF states of pacemaker's actuator ($PM_Actuator_A$, $PM_Actuator_V$) of each chamber (atria and ventricular) and also control the sensing intrinsic stimulus of ventricular chamber and synchronizes ON and OFF states of pacemaker's sensor (PM_Sensor_A, PM_Sensor_V) in atrial and ventricular chambers under real time constraints. We modify the guard ($grd1$) of this event and add more properties to synchronize the pacing and sensing activities and we also add new actions ($act2$ and $act3$). The additional guards and action handle the behavior of events (Thr_A_val and Thr_V_val) to sense the intrinsic activities from the atrial and ventricular chambers.

EVENT tic_AV
WHEN
⊕ grd4 $PM_Sensor_V = ON$
⊕ grd5 $Thr_V_State = FALSE$
⊕ grd6 $Thr_V < STA_THR_V$
⊕ grd7 $PM_Actuator_V = OFF$
⊕ grd8 $PM_Actuator_A = OFF$
THEN
⊕ act3 : $Thr_V_State := TRUE$
END

We add some new guards ($grd4 - grd8$) and an action ($act3$) in event (tic_AV) of this refinement. The new guards provide more specific and stronger guards to count the atrioventricular (AV) interval and action ($act3$) states that threashold state (Thr_V_State) of ventricular sets TRUE.

First refinement of VDD mode:

In the first refinement of VDD operating mode, we formalize the concept of sensing threshold value of the double electrode pacemaker. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of heart and a pulse generator for deliverying stimulation pulses to the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value under safety margin. We introduce the new variables (Thr_A and Thr_V) to hold the sensing threashold value of the pacemaker's sensor (PM_Sensor_A, PM_Sensor_V) of atrial and ventricular chambers. Similarly next variables (Thr_A_State and Thr_V_State) represent TRUE or FALSE state of the pacemaker's sensor (PM_Sensor_A, PM_Sensor_V) to sense the intrinsic activity of the atrial and ventricular chambers.

$inv1 : Thr_A \in \mathbb{N}_1$
$inv2 : Thr_V \in \mathbb{N}_1$
$inv3 : Thr_A_State \in BOOL$
$inv4 : Thr_V_State \in BOOL$
$inv5 : sp > VRP \wedge sp < Pace_Int \wedge AV_Count_STATE = FALSE \Rightarrow PM_Sensor_A = ON$
$inv6 : PM_Actuator_V = ON \Rightarrow (sp = Pace_Int) \vee sp < Pace_Int \wedge AV_Count > V_Blank \wedge AV_Count \geq FixedAV$
$inv7 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge AV_Count_STATE = TRUE \Rightarrow PM_Sensor_A = OFF$
$inv8 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge AV_Count_STATE = TRUE \Rightarrow PM_Sensor_V = ON$

From invariants ($inv5 - inv8$) represent the safety properties of the pacemaker system under pacing and sensing activities of electrode in VDD operating mode. The fifth invariant ($inv5$) states that the pacemaker's sensor (PM_Sensor_A) of atrial is ON when current clock counter sp is greater than VRP and less than pace interval ($Pace_Int$) and atrioventricular (AV) counter state (AV_Count_STATE) is FALSE. The next invariant ($inv6$) represents that the pacemaker's actuator ($PM_Actuator_V$) of ventricular is ON when either the current clock counter sp is equal to the pace interval ($Pace_Int$) or clock counter sp is less then pace interval ($Pace_Int$), atrioventricular counter (AV_Count) is greater than blanking period (V_Blank) and greater than or equal to the atrioventricular (AV) interval ($FixedAV$). The last two invariants ($inv7$, $inv8$) represent that the pacemaker's sensor (PM_Sensor_A) of atrial is OFF and the pacemaker's sensor (PM_Sensor_V) of ventricular is ON when the current clock counter sp is greater than ventriculoatrial (VA) interval, less than pace interval ($Pace_Int$) and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE.

EVENT Thr_Value_V
WHEN
grd1 : $Thr_V_val \in \mathbb{N}$
grd2 : $PM_Sensor_V = ON$
grd3 : $Thr_V_State = TRUE$
grd4 : $Thr_V < STA_THR_V$
grd5 : $(sp \geq VRP \wedge sp < Pace_Int - FixedAV)$
$\vee$
$(sp \geq Pace_Int - FixedAV \wedge sp < Pace_Int)$
grd6 $(Thr_A_State = FALSE \wedge Thr_A < STA_THR_A)$
$\vee$
$(PM_Sensor_A = OFF \wedge AV_Count < FixedAV)$
THEN
act1 : $Thr_V := Thr_V_val$
act2 : $Thr_V_State := FALSE$
END

In this refinement we introduce the two new events (Thr_Value_V and Thr_Value_A) for sensing the intrinsic activities from ventricular and atrial chambers. These events are synchronized with all other events of this operating mode under all safety properties and real time constraints. The guards ($grd2 - grd4$) of event (Thr_Value_V) state that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON, threashold state (Thr_V_State) of ventricular is TRUE and sensed value (Thr_V) is less than standard threshold value (STA_THR_V) of ventricular chamber. The next guard ($grd5$) represents that either clock counter (sp) is greater then and equal to VRP and less than ventriculoatrial (VA) interval or clock counter (sp) is greater than or equal to atrioventricular (AV) interval and less then pace interval ($Pace_Int$). The last guard ($grd6$) states that either threshold state (Thr_A_State) of atrial chamber is FLASE and threashold value (Thr_A) of atrial is less then standard threashold value (STA_THR_A) of atrial chamber or pacemaker's sensor PM_Sensor_A of atrial is OFF and atrioventricular (AV) counter (AV_Count) is less than atrioventricular (AV) interval ($FixedAV$). The actions ($act1 - act2$) of this event state that actual sensed value (Thr_V_val) of ventricular chamber assigns to variable (Thr_V) and sets FALSE state of threashold ventricular state (Thr_V_State).

EVENT Thr_Value_A
WHEN
grd1 : $Thr_A_val \in \mathbb{N}$
grd2 : $PM_Sensor_A = ON$
grd3 : $Thr_A_State = TRUE$
grd4 : $Thr_A < STA_THR_A$
grd5 : $(sp \geq VRP \wedge sp < Pace_Int)$
THEN
act1 : $Thr_A := Thr_A_val$
act2 : $Thr_A_State := FALSE$
END

Other new event (Thr_Value_A) introduce to take the intrinsic activities of atrial chamber. The guards ($grd2 - grd4$) state that the pacemaker's sensor (PM_Sensor_A) of atrial chamber is ON, threashold state (Thr_A_State) of atrial chamber is TRUE and sensed value (Thr_A) of atrial chamber is less than standard threashold (STA_THR_A) of atrial chamber. The last guard of this event state that the current clock counter sp is greater than or equal to VRP and less than pace interval ($Pace_Int$) . The actions ($act1 - act2$) of this event state that actual sensed value (Thr_A_val) of atrial chamber assigns to variable (Thr_A) and sets FALSE state of threashold atrial state (Thr_A_State).

EVENT Actuator_OFF_V
⊕ act6 : $Thr_A := 0$
⊕ act7 : $Thr_V := 0$
⊕ act8 : $Thr_A_State := FALSE$
⊕ act9 : $Thr_V_State := FALSE$

EVENT Sensor_ON_A
⊕ act2 : $Thr_A_State := TRUE$

EVENT Sensor_OFF_A
⊕ grd3 : $Thr_A \geq STA_THR_A$
⊕ act2 : $Thr_A_State := TRUE$

EVENT Sensor_OFF_V
⊕ grd5 : $Thr_V \geq STA_THR_V$
⊕ act7 : $Thr_A := 0$
⊕ act8 : $Thr_V := 0$
⊕ act9 : $Thr_A_State := FALSE$
⊕ act10 : $Thr_V_State := FALSE$

We add some new actions and guards in events ($Actuator_OFF_V$, $Sensor_ON_A$, $Sensor_OFF_A$, and $Sensor_OFF_V$), to synchronize the sensing activities using events (Thr_A_val and Thr_V_val) under real time constraints, which are already defined in the abstract model of this operating mode.

EVENT tic
WHEN
grd1 : $(sp < VRP$
$\lor$
$(sp \geq VRP \land sp < Pace_Int - FixedAV \land$
$PM_Sensor_V = ON \land PM_Sensor_A = ON \land$
$Thr_V_State = FALSE \land Thr_V < STA_THR_V))$
grd2 : $AV_Count_STATE = FALSE$
THEN
⊕ act2 : $Thr_A_State := TRUE$
⊕ act3 : $Thr_V_State := TRUE$
END

The event (tic) of this refinement model progressively increases the current clock counter sp under predefined pace interval ($Pace_Int$). The guard of this event controls the pacing stimulus into the heart chambers (atria and ventricular), synchronizes ON and OFF states of the pacemaker's actuator ($PM_Actuator_V$) of ventricular chamber and also control the sensing intrinsic stimulus of atrail and ventricular chambers and synchronizes ON and OFF states of the pacemaker's sensors (PM_Sensor_A, PM_Sensor_V) in heart chambers under real time constraints. We modify the guard ($grd1$) and new guard ($grd2$) of this event and add more properties to synchronize the pacing and sensing activities and we also add new actions ($act2$ and $act3$). The additional guards and action handle the behavior of events (Thr_A_val and Thr_V_val) to sense the intrinsic activities from the atrial and ventricular chambers.

EVENT tic_AV
  WHEN
⊕ grd4 $PM_Sensor_V = ON$
⊕ grd5 $Thr_V_State = FALSE$
⊕ grd6 $Thr_V < STA_THR_V$
⊕ grd7 $PM_Actuator_V = OFF$
⊕ grd8 $PM_Sensor_A = OFF$
  THEN
⊕ act3 : $Thr_V_State := TRUE$
  END

We add some new guards ($grd4 - grd8$) and an action ($act3$) of this event (tic_AV) in refinement. The new guards provide more specific and stronger guards to count the atrioventricular (AV) interval and action ($act3$) states that threashold state (Thr_V_State) of ventricular sets TRUE.

First refinement of DDD mode:

In the first refinement of DDD operating mode, we formalize the concept of sensing threshold value of the double electrode pacemaker. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of heart and a pulse generator for deliverying stimulation pulses to the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value under safety margin. We introduce the new variables (Thr_A and Thr_V) to hold the sensing threashold value of the pacemaker's sensor (PM_Sensor_A, PM_Sensor_V) of atrial and ventricular chambers. Similarly next variables (Thr_A_State and Thr_V_State) represent TRUE and FALSE states of the pacemaker's sensor (PM_Sensor_A, PM_Sensor_V) to sense the intrinsic activity of the atrial and ventricular chambers.

$inv1 : Thr_A \in \mathbb{N}_1$
$inv2 : Thr_V \in \mathbb{N}_1$
$inv3 : Thr_A_State \in BOOL$
$inv4 : Thr_V_State \in BOOL$
$inv5 : sp > VRP \wedge sp < Pace_Int - FixedAV \Rightarrow PM_Sensor_V = ON$
$inv6 : PM_Actuator_V = ON \Rightarrow (sp = Pace_Int) \vee$
  $sp < Pace_Int \wedge AV_Count > V_Blank \wedge AV_Count \geq FixedAV$
$inv7 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge$
  $AV_Count_STATE = TRUE \Rightarrow PM_Sensor_A = OFF$
$inv8 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge$
  $AV_Count_STATE = TRUE \Rightarrow PM_Sensor_V = ON$
$inv9 : sp > Pace_Int - FixedAV \wedge sp < Pace_Int \wedge$
  $AV_Count_STATE = TRUE \Rightarrow PM_Actuator_A = OFF$
$inv10 : PM_Actuator_A = ON \Rightarrow sp \geq Pace_Int - FixedAV$

From invariants ($inv5 - inv10$) represent the safety properties of the pacemaker system under pacing and sensing activities of electrode in DDD operating mode. The fifth invariant ($inv5$) states that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON when current clock counter sp is greater than VRP and less than ventriculoatrial (VA) interval. The next invariant ($inv6$) represents that the pacemaker's actuator ($PM_Actuator_V$) of ventricular is ON when either the current clock counter sp is equal to the pace interval ($Pace_Int$) or clock counter sp is less then pace interval ($Pace_Int$), atrioventricular counter (AV_Count) is greater than blanking period (V_Blank) and greater than or equal to the atrioventricular (AV) interval ($FixedAV$). The next three invariants ($inv7$, $inv8$, $inv9$) represent that the pacemaker's sensor (PM_Sensor_A) of atrial is OFF, the pacemaker's sensor (PM_Sensor_V) of ventricular is ON and the pacemaker's actuator ($PM_Actuator_A$) of atrial is OFF, when current clock counter sp is greater than ventriculoatrial (VA) interval, less than pace interval ($Pace_Int$) and atrioventricular (AV) counter state (AV_Count_STATE) is TRUE. The last invariant states that the pacemaker's actuator of atrial

chamber is ON when current clock counter sp is greater than or equal to the ventriculoatrial (VA) interval $(Pace_Int - FixedAV)$.

EVENT Thr_Value_V
WHEN
grd1 : $Thr_V_val \in \mathbb{N}$
grd2 : $PM_Sensor_V = ON$
grd3 : $Thr_V_State = TRUE$
grd4 : $Thr_V < STA_THR_V$
grd5 : $(sp \geq VRP \land sp < Pace_Int - FixedAV)$
$\lor$
$(sp \geq Pace_Int - FixedAV \land sp < Pace_Int)$
grd6 $(Thr_A_State = FALSE \land Thr_A < STA_THR_A)$
$\lor$
$(PM_Sensor_A = OFF \land AV_Count < FixedAV)$
THEN
act1 : $Thr_V := Thr_V_val$
act2 : $Thr_V_State := FALSE$
END

In this refinement, we introduce the two new events (Thr_Value_V and Thr_Value_A) for sensing the intrinsic activities from ventricular and atrial chambers. These events are synchronized with all other events of this operating mode under all safety properties and real time constraints. The guards $(grd2 - grd4)$ of event (Thr_Value_V) state that the pacemaker's sensor (PM_Sensor_V) of ventricular is ON, threashold state (Thr_V_State) of ventricular is TRUE and sensed value (Thr_V) is less than standard threshold value (STA_THR_V) of the ventricular chamber. The next guard $(grd5)$ represents that either clock counter sp is greater then and equal to VRP and less than ventriculoatrial (VA) interval or clock counter sp is greater than or equal to atrioventricular (AV) interval and less then pace interval $(Pace_Int)$. The last guard $(grd6)$ states that either threshold state (Thr_A_State) of atrial is FLASE and threashold value (Thr_A) of atrial is less then standard threashold value (STA_THR_A) of atrial chamber or the pacemaker's sensor (PM_Sensor_A) of atrial chamber is OFF and atrioventricular (AV) counter is less than atrioventricular (AV) interval $(FixedAV)$. The actions $(act1 - act2)$ of this event state that actual sensed value (Thr_V_val) of ventricular chamber assigns to variable (Thr_V) and sets the FALSE state of threashold ventricular state (Thr_V_State).

EVENT Thr_Value_A
WHEN
grd1 : $Thr_A_val \in \mathbb{N}$
grd2 : $PM_Sensor_A = ON$
grd3 : $Thr_A_State = TRUE$
grd4 : $Thr_A < STA_THR_A$
grd5 : $(sp \geq VRP \land sp < Pace_Int - FixedAV)$
THEN
act1 : $Thr_A := Thr_A_val$
act2 : $Thr_A_State := FALSE$
END

Other new event (Thr_Value_A) introduce to take the intrinsic activities of atrial chamber. The guards $(grd2 - grd4)$ state that pacemaker's sensor (PM_Sensor_A) of atrial chamber is ON, threashold state (Thr_A_State) of atrial chamber is TRUE and sensed value (Thr_A) of atrial chamber is less than standard threashold (STA_THR_A) of atrial chamber. The last guard of this event state that clock counter sp is greater than or equal to VRP and less than ventriculoatrial (VA) interval. The actions $(act1 - act2)$ of this event state that actual sensed value (Thr_A_val) of atrial chamber assigns to variable (Thr_A) and sets FALSE state of threashold atrial state (Thr_A_State).

EVENT Actuator_OFF_V
⊕ act6 : $Thr_A := 0$
⊕ act7 : $Thr_V := 0$
⊕ act8 : $Thr_A_State := FALSE$
⊕ act9 : $Thr_V_State := FALSE$

EVENT Sensor_ON_A
⊕ act2 : $Thr_A_State := TRUE$

EVENT Sensor_OFF_A
⊕ grd3 : $Thr_A \geq STA_THR_A$
EVENT Sensor_OFF_V
⊕ grd6 : $Thr_V \geq STA_THR_V$
⊕ act7 : $Thr_A := 0$
⊕ act8 : $Thr_V := 0$
⊕ act9 : $Thr_A_State := FALSE$
⊕ act10 : $Thr_V_State := FALSE$

We add some new actions and guards in events ($Actuator_OFF_V$, $Sensor_ON_A$, $Sensor_OFF_A$, and $Sensor_OFF_V$) to synchronize the sensing activities using events (Thr_A_val and Thr_V_val) under real time constraints, which are already defined in the abstract model of this operating mode.

EVENT tic
WHEN
grd1 : $(sp < VRP \land AV_Count_STATE = FALSE$
$\lor$
$(sp \geq VRP \land sp < Pace_Int - FixedAV \land$
$PM_Sensor_V = ON \land PM_Sensor_A = ON \land$
$Thr_V_State = FALSE \land Thr_V < STA_THR_V))$
grd2 : $AV_Count_STATE = FALSE$
THEN
⊕ act2 : $Thr_A_State := TRUE$
⊕ act3 : $Thr_V_State := TRUE$
END

The event (tic) of this refinement model progressively increases the current clock counter sp under predefined pace interval ($Pace_Int$). The guard ($grd1$) of this event control the pacing stimulus into the heart chambers (atria and ventricular), synchronizes ON and OFF states of the pacemaker's actuator ($PM_Actuator_A$, $PM_Actuator_V$ of each chamber (atria and ventricular) and also control the sensing intrinsic stimulus of atrial and ventricular chambers and synchronizes ON and OFF states of the pacemaker's sensor (PM_Sensor_A, PM_Sensor_V) of atrial and ventricular under real time constraints. We modify the guard ($grd1$) and new guard ($grd2$) of this event and add more properties to synchronize the pacing and sensing activities and we also add new actions ($act2$ and $act3$). The additional guards and action handle the behavior of events (Thr_A_val and Thr_V_val) to sense the intrinsic activities from the atrial and ventricular chambers.

EVENT tic_AV
WHEN
⊕ grd4 $PM_Sensor_V = ON$
⊕ grd5 $Thr_V_State = FALSE$
⊕ grd6 $Thr_V < STA_THR_V$
⊕ grd7 $PM_Actuator_V = OFF$
⊕ grd8 $PM_Sensor_A = OFF$
⊕ grd9 $PM_Actuator_A = OFF$
THEN
⊕ act3 : $Thr_V_State := TRUE$
END

We add some new guards $(grd4 - grd9)$ and an action $(act3)$ of event (tic_AV) in this refinement. The new guards provide more specific and stronger guards to count the atrioventricular (AV) interval and action $(act3)$ states that threashold state (Thr_V_State) of ventricular sets TRUE.

5.4.3 Second refinement:Rate Modulation

Rate modulation term is used to describe the capacity of a pacing system to respond to physiologic need by increasing and decreasing pacing rate. The rate modulation mode of the pacemaker can progressively pace faster than the lower rate, but no more than the upper sensor rate limit, when it determines that heart rate needs to increase. This typically occurs with exercise in patients that cannot increase their own heart rate. The amount of rate increase is determined by the pacemaker on the basis of maximum exertion is performed by the patient. This increased pacing rate is sometimes referred to as the "sensor indicated rate". When exertion has stopped the pacemaker will progressively decrease the paced rate down to the lower rate.

In this final refinement, we introduce the rate modulation function and found some new operating modes (AOOR,VOOR,AAIR,VVIR,AATR and VVTR) of the pacemaker system. For modeling the rate modulation, we introduce the new constants maximum sensor rate MSR as $MSR \in 50 .. 175$ and acc_thr as $acc_thr \in \mathbb{N}$. The maximum sensor rate (MSR) is the maximum pacing rate allowed as a result of sensor control and it must be between 50 and 175 pulse per minute (ppm). The constant acc_thr represents the activity threshold. A new variable $acler_sensed$ is defined as $acler_sensed \in \mathbb{N}$, to store the measured value from the accelerometer. The accelerometer is used to measure the physical activities of the body in a pacemaker system.

The two invariants $(inv1, inv2)$ provide the safety margin and state that the heart rate never falls below the lower rate limit (LRL) and never exceed the maximum sensor rate (MSR) limit.

inv1 $acler_sensed < acc_thr$
$\Rightarrow$
$Pace_Int = 60000/LRL$

inv2 $acler_sensed > acc_thr$
$\Rightarrow$
$Pace_Int = 60000/MSR$

In this final refinement, we introduce only two new events $Increase_Interval$ and $Decrease_Interval$, to control the pacing rate of the double electrode pacemaker in the rate modulating operating modes. The new events $Increase_Interval$ and $Decrease_Interval$ control the value of pace interval variable $Pace_Int$, whenever a measured value ($acler_sensed$) from the accelerometer sensor goes higher or lower than the activity threshold acc_thr.

EVENT Increase_Interval
  WHEN
    grd1 $acler_sensed > acc_thr$
  THEN
    act1 $Pace_Int := 60000/MSR$
  END

EVENT Decrease_Interval
  WHEN
    grd1 $acler_sensed < acc_thr$
  THEN
    act1 $Pace_Int := 60000/LRL$
  END

Finally, we have modeled all the functional and parametric requirements of different operating modes of the double electrode pacemaker system using stepwise refinements. We have also discovered the hierarchical development and relationship among all operating modes (see Figure-4) of double electrode cardiac pacemaker.

5.5 Model Validation and Analysis

There are two main validation activities in EVENT B and both are complementary for designing a consistent system:

- *consistency checking*, which is used to show that the events of a machine preserve the invariant, and *refinement checking*, which is used to show that one machine is a valid refinement of another. A list of automatically generated proof obligations should be discharged by the proof tool of the RODIN platform.
- *model analysis*, which is done by the ProB tool and consists in exploring traces or scenarios of our consistent EVENT B models. For instance, the ProB may discover possible deadlocks or hidden properties that are not expressed by generated proof obligations.

This section conveys the validity of the model by using ProB tool [89, 83] and Proof Statistics. "Validation" refers to the activity of gaining confidence that the developed formal models are consistent with the requirements, which expressed in the requirements document [68]. We have used the ProB tool [89] that supports *automated consistency checking* of EVENT B machines via model checking [72] and constraint-based checking [80]. Animation using ProB worked very well and we have then used ProB to validate the EVENT B machine. This tool assists us to find potential problems, to improve invariants expressions in our EVENT B models, for instance by generating counter-examples when it discovers an invariant violation. ProB may help in improving invariant expression by suggesting hints for strengthening the invariant and each time an invariant is modified, new proof obligtions are generated by the RODIN platform. It is the complementary use of both techniques to develop formal models of critical systems, where high safety and security are required. More errors are corrected during the elaboration of the specifications while discharging the proof obligations and careful cross-reading than during the animations. We have validated all operating modes of the pacemaker in each refinement of models. The pacemaker specification is developed and formally proved by the RODIN tool.
ProB was very useful in the development of the pacemaker specification, and was able to animate all of our models and able to prove the absence of error (no counter example exist). The ProB model checker also discovered several invariant violations, e.g., related to incorrect responses or unordered pacing and sensing activities. It was also able to discover a deadlock in two of the models, which was due to the fact that "clock counter" were not properly recycled, meaning that after a while no pacing or sensing activities occur into the system. Such kind of errors would have been more difficult to uncover with the prover of RODIN tool.

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	166	125(76%)	41(24%)
First Refinement	211	190(90%)	21(10%)
Second Refinement	67	66(99%)	1(1%)
Total	444	381(86%)	63(14%)

Table-3 : Proof statistics

The Table-3 is expressing the proof statistics of the development in the RODIN tool. These statistics measure the size of the model, the proof obligations generated and discharged by the RODIN prover, and those are interactively proved. The complete development of double electrode pacemaker system results in 444(100%) proof obligations, in which 381(86%) are proved automatically by the RODIN tool. The remaining 63(14%) proof obligations are proved interactively using RODIN tool. In the model, many proof obligations are generated due to the introduction of new functional behaviors and their parameters (threshold, hysteresis and rate modulation) under real-time constraints. In order to guarantee the correctness of these functional behaviors, we have established various invariants in stepwise refinement. Most of the proofs are interactively discharged in the abstract level and the 1st refinement. These proof are quite simple, and achieve with the help of *"do case"* and instantiation of the constants and variables. The guards of some events are very complex, so for proving the invariants and the theorems, we simplify the guards using *"do case"*. The first abstract level is in detailed due to introduction of pacemaker's actuators and sensors of two electrodes for both heart chambers with proper synchronized properties under real time constraints.

5.6 Conclusion and Future Works

In this report, we have presented the formal specification of the double electrode pacemaker, is a grand challenge that is proposed by the Verified Software Initiative. We have used the EVENT B formal language on RODIN platform to develop the formal model of the operating modes of the double electrode pacemaker. Our approach for formalizing and reasoning about functional behaviour of pacing and sensing activities of the pacemaker based on action-reaction under real-time constraints.
The pacemaker case study suggests that such an approach can yield a viable model that can be subjected to useful validation against system-level properties at an early stage in the development process. We have applied the action-reaction [59] and time based patterns [70, 91] to develop the pacemaker system. The proposed techniques based on development patterns intend to assist in the design process of system where correctness and safety are important issues.
More precisely, we have presented development of operating modes of double electrode pacemaker system. For quick understanding, we have formalized several different developments, each highlighting a different aspect of the problem, making different assumptions about the operating modes and establishing different properties. For example, we have considered a case of constant pacing in both chanbers, sensing and pacing synchronously, threshold parameter for an electrode sensor and rate modulation operating modes. In a stepwise refinement, we have also discovered the hierarchical development and relationship among double electrodes operating modes of the pacemaker (see Fig. 4).
Our developments reflect not only the many facets of the problem, but also that there is a learning process involved in understanding the problem and its ultimate possible solutions. The approach is concerned with separation : firstly, it proves the basic behavior of double electrode pacemaker system at abstract level secondly it introduces the peculiarity of the specific properties. We have proved the fundamental properties in the beginning, namely the action-reaction with real-time constraints and the uniqueness of a solution, are kept through the refinement process (provided, of course, the required proofs are done).
The consistency of our specification has been checked through mathematical reasoning and validation experiments are performed by ProB model checker regarding safety conditions. As part of our reasoning, we have proved that the initialisation of the system is a valid one and we have calculated the preconditions of the operations. The latter has been executed to guarantee that our intention to have total operations has been fulfilled. At every stage of refinement we introduced the new parametric functionality of the system and proved the *consistency* and *refinement checking*. We introduced the invariants at refinement level that the initialisation of the whole system is valid. Proofs were quite simple, and achieved with the help of *instantiation* and *"do case"*. The guards of some events are very complex, so for proving the invariants, we simplify the guards using *"do case"*. In total, we have proved 444(100%) proof obligations, in which

381(86%) are proved automatically and remaining 63(14%) proof obligations are proved interactively using RODIN tool (see Table-3). Finally, we have validated the double electrode pacemaker system using the ProB model checker as validation tool and verify the correctness of our proved double electrode pacemaker system under the real-time constraints.
In the future, we have planned to create a formal proof based simulator for the single and double electrode pacemaker. It can be used by the doctor to analyze the real-time heart signal and predict the operating modes. As far as, it can be also used as a diagnostic tool to diagnose the patient and help to take the better decision for implanting a pacemaker [88]. For our on going research, we have contacted with physician and cardiologist experts to generalized the operating modes of single and double electrode pacemaker and derive the common parametric functional properties through refinement tree structures that help to design the automatic mode switching from one operating mode to another operating mode according to the heart pacing requirement.

Chapter 6

Adhoc systems

Sommaire

6.1 Introduction 110
6.2 Overview of the modelling protocols 111
6.3 Abstract model of basic communication protocol 113
6.3.1 First Refinement 116
6.3.2 Second Refinement 118
6.3.3 Third Refinement : Route Discovery Protocol 121
6.3.4 Fourth Refinement : Route Discovery Protocol 122
6.3.5 Fifth Refinement : Route Discovery Protocol 124
6.4 Conclusion 126

Test and Simulation are the only verification techniques used for ad-hoc network routing protocol. Although these techniques give us an excellent overview of the protocol behaviour, some undesirable aspects of the protocol could still be undiscovered. Therefore formal verification is needed. This paper presents the incremental development of the Dynamic Source Routing (DSR) ad-hoc wireless network protocol using underlying modelling language Event-B and its associated proof tools. The development is performed in a stepwise manner composing more advanced routing components between the abstract specification of the ad-hoc network wireless protocol and topology was verified through a series of refinements.This model developed in two phases, first phase for basic communication protocol and second phase for route discovery protocol. Our contribution are in this paper to model the adhoc wireless network protocol using Event-B and prove it.The initial model of the ad-hoc network wireless protocol provides the basic properties of the graph (acyclic, connectivity, symmetric) and in abstraction of this model contain the five events of basic communication protocol. In next stages of refinements include the detail information and more events are introduced. The final step of refinement completely localized the events and similar to implementation of ad-hoc network wireless protocol. The stepwise refinement of the ad-hoc network wireless protocol help us to achieve a high degree of automatic proof.

6.1 Introduction

An ad-hoc network is a collection of wireless mobile nodes forming a temporary network without any established network or centralized administration with the capability to communicate with each other. Every mobile node acts as a router and forward traffic originated by other nodes. Each node is able to dynamically discover and maintain routes to other node in the network. Established routes should be loop-free and route changes should coverage quickly in large networks. The routing problem in a real ad-hoc network is complicated due to non-uniform propagation characteristics of wireless transmission and movement of nodes at any time [9].

There are various type of ad-hoc network wireless protocol. This paper described the incremental development of the Dynamic Source Routing (DSR) ad-hoc wireless network protocol. In Source routing technique the sender determine the complete sequence of nodes through which forward the data packet. The sender explicitly lists this route in the packets header, identifying each forwarding 'hop' by the address of the next node to which, transmit the data packet on its way to the destination node. The protocol presented here is explicitly designed for use in the wireless environment of an ad-hoc network. There are no periodic router advertisements in the protocol. Instead, when a node needs a route to another node, it dynamically determines one based on local routing table or route cached information and on the results of a route discovery protocol [9].

The conventional routing protocols are not designed for the type of dynamic topology changes that may be present in ad-hoc networks. In conventional networks, links between routers occasionally go down or come up, and sometimes the cost of a link may change due to congestion, but routers do not generally move around dynamically. In an environment with mobile nodes as routers, though, convergence to new, stable routes after such dynamic changes in network topology may be slow, particularly with distance vector algorithms. Our dynamic source routing protocol is able to adapt quickly changes such as node movement, yet requires no routing protocol overhead during periods in which such changes do not occur.

Validation techniques used for ad-hoc network protocols are only simulation and testing. This is an operational way to check weather a given system realization confirm to an abstract specification. By nature, testing can be applied only after a prototype implementation of the system has been realized. Formal verification, as opposed to testing, work on models (rather than implementation) and amounts to mathematical proof of correctness of a system. We present a way to model and prove ad-hoc network wireless protocols using Event-B as a modelling language. Proving properties on ad-hoc wireless network protocols in dynamic environment is a challenging task due to continue changing the network routes. Only a complementary technique to simulation and testing is to prove that a system operates correctly. The term for this mathematical demonstration of the correctness of a system is formal verification. In model checking or theorem proving, algorithms executed by computer tools are used in order to verify the correctness of systems. The user gives the description of system and defines the requirements. Knowing these parameters the machine can perform a verification of a model. The user can refine the model until the model specifications converge to the real system.

The basic requirements of the ad-hoc network system are basic communication protocol and route discovery

protocol.So we have introduced the two phases of ad-hoc network model.The basic communication protocol is the first phase and route discovery protocol is second phase of our ad-hoc network model.The abstract specification of route discovery protocol defined under the first phase of the model or in other way we can say that it is an event of basic communication protocol, which is triggered in a specific condition. In the development of ad-hoc network system we proposed the following idea, how to model the integration of sub system of same domain and each subsystem of the system has different functionality and main system of the domain is dependent on subsystem for some activities.
Our abstract specification includes events modelling atomic transfer of data packets between moving nodes, loss of data packets from the route, successfully receiving of data packets by destination node and routes are changing due to movement in nodes. The safety property of basic communication protocol is represented as total number of sending data packets should be equal to received and lost data packets in the system. The nature of the refinement that we verified using RODIN proof tools are safety refinement and any behaviour (trace of events) of a refined model must be behaviours of the abstract model. Thus, since a behaviour which results in the transferring of data packets from source node to destination node and route updating is preserving by the abstract model, it is also preserved in a correctly refined models. Our refinements break the atomicity of transferring the data packets into several small process or events. In event of sending the data packets can be aborted due to network failure (intermediate node failure or not availability of any route from source node to destination node). In case of data packet is not received by destination node, the source node will re-send the data packet after a specified time interval known as timeout function.

6.2 Overview of the modelling protocols

The ad-hoc network wireless protocol is defined by a proof-based development of Event-B models which are modelling techniques in a very abstract and general way. The wireless ad-hoc routing protocol maintain up-to-date routes between all nodes in the network with routing table and link-update propagations. If there are still no valid routes for a specific destination, it uses a route discovery process. Here we will just present a sufficient overview of the abstract specification and refinement stages in order to help the reader understand the rational of each refinement.

Abstract model

In the abstract model of stepwise development contains definition and properties of the network (g), finite set of nodes (ND) and definition of closure for checking the connectivity of two distinct nodes in the network. The first model contains the five basic events $sending$, $receiving$, $losing$, $remove_link$ and add_link, which are elementary events of basic communication protocol in ad-hoc network.

First refinement

In the first refinement, we add the more detail information of the communication protocol.When data packet passes by all intermediate nodes, the intermediate nodes are using the store and forward architecture. In this refinement we introduced the variable $(gstore)$ which store the data packets at every intermediate node, when an intermediate node forward the data packet to next neighboring node of the route using event $(forward)$.

Second refinement

This refinement is relatively complex refinement in which we introduced the route cache concept at every node of the ad-hoc network.If the source node wants to send the data packet to any destination node, first of all it will check own routing table and if it is unable to find the route from the route cache or local routing table then it will start to discover the new route for the destination node using route discovery protocol. The event $(update_routing_table)$ is updating the route cache of all intermediate nodes from source node to destination node.This event is also an abstraction of second phase of ad-hoc network as route discovery protocol.

Third refinement

In this refinement, we have introduced the new events $(broadcast_rrq)$ and $(received_rrq)$ for discovering the new routes.The event $(broadcast_rrq)$ send the route request packet, which may be received by those nodes within wireless transmission range of it.The route request packet identifies the node, referred to as the destination node of the route discovery, for which route is requested.If the route discovery is successful the source node receives a route reply packet listing a sequence of network hops through which it may reach the destination node using the $(received_rrq)$ event.

Fourth refinement

Table 6.1: Proof obligation of the ad-hoc network wireless protocol

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	16	16(100%)	0(0%)
First Refinement	37	20(55%)	17(45%)
Second Refinement	12	11(91%)	1(9%)
Third Refinement	5	5(100%)	0(0%)
Fourth Refinement	19	17(89%)	2(11%)
Fifth Refinement	12	12(100%)	0(0%)
Total	101	81(80%)	20(20%)

In addition to the address of the original initiator of the request and the target of the request, each route request packet contains a route record, in which is accumulated a record of the sequence of hops taken by the route request packet as it is propagated through the ad-hoc network during this route discovery.In this refinement of the system we have introduced a new event ($forward_rrq$).The new variable ($route_record_rrq$) used to store the link information at the time of propagation of route request packets from one node to other node.If the route request receiver node is not the target node then it add the link information to the route record of route request packet and again broadcast it.

Fifth refinement

In this refinement of the route discovery protocol we introduced the architecture of storage of route record at every node. Each route request packet also contains a unique request id, set by the initiator from a locally-maintained sequence number. In order to detect duplicate route requests received, each host in the ad hoc network maintains a list of route request packet that it has recently received on any route request. The route request thus propagates through the ad hoc network until it reaches the target host, which then replies to the initiator. The original route request packet is received only by those hosts within wireless transmission range of the initiating host, and each of these hosts propagates the request if it is not the target and if the request does not appear to this host to be redundant. Discarding the request because the host's address is already listed in the route record guarantees that no single copy of the request can propagate around a loop. Also discarding the request when the host has recently seen one with the same route request packet removes later copies of the request that arrive at this node by a different route.

Through careful use of small refinement steps and appropriate intermediate abstractions, we were able to achieve an impressive degree of automatic proof. Here we have mentioned the table of proof obligations for ad-hoc network.

All the proof obligations for all six levels were generated and proved using the RODIN proof tool. The statistics from the mechanical proof effort for all of the refinement levels are outlined in **Table-1**. In the table, the Total number of POs column represents the total number of proof obligations generated for each level. The Interactive Proof column represents the number of those proof obligations that had to be proved interactively. Those proof obligations that were not proved interactively were proved completely automatically by the prover.

The complete development of ad-hoc network protocol resulted in 101 proof obligations, in which 81(80%) were proved completely automatically by RODIN tool. The remaining 20(20%) proof obligations were proved interactively using RODIN tool. This refinement approach together with the RODIN tool supports an incremental style of system development. We have presented the complete refinements in top down manner. We started with the highest level specification and then produced a model approximating the lowest level. However in attempting to prove refinement between these models it was clear that the abstraction gap was too large would have required a complex gluing invariant. Instead we decided that some intermediate abstraction was required. Any modifications to the refinement model had an impact on the existing proofs. For that we have need to give the proper gluing invariants.

As we can see from the **Table-1** in refinement first we have proved 17 proofs obligations interactively out of

total no. of 37 proofs obligations.The most difficult proof was in first and fourth refinement of the abstract model. Most of the interactive proof generated due to continuous changing of the network and maintain the connectivity using transitive closure properties in dynamic environment. We know that network is not fixed, all nodes are moving. So network structure are continuously changing. Some old connectivity are getting lost and some new connectivity is forming in the current network due to movement of the nodes. So proof of this dynamic property of the network is quite difficult rather then other properties of the network protocol. So the instantiation had to be done manually in interactive prover for proving the transitive closure property in dynamic environment with the help of appropriate axioms.In the fourth refinement we have added two extra invariants for poving the proof obligation in route discovery protocol.The complete details are avaialble in fourth refinement.

6.3 Abstract model of basic communication protocol

The basic block diagram of ad-hoc network protocol is shown in Figure-1.In this figure six nodes are randomly distributed and source node (A) want to send the data packets to destination node (D). The straight line represents the connectivity of ad-hoc network and dash line represents the route from source node (A) to destination node (D).

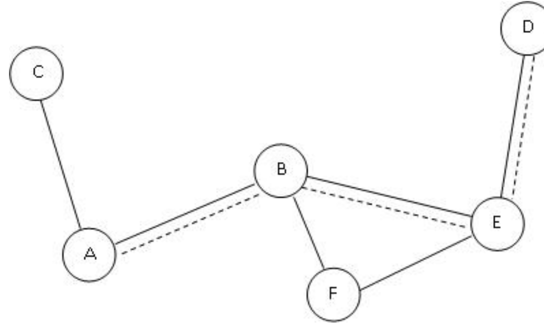


Figure 6.1: Ad-hoc Networks

We suppose that an ad-hoc network wireless protocol allows a set of users to exchange the data packets among themselves. Each user reside at a mobile node, and each user may engage in either sending or receiving actions. Our abstract B model of the ad-hoc system introduces: a set of moving nodes ND with following assumptions:

1. The network is supported by a graph (g) built on (ND).
2. The links between the nodes are bidirectional, meaning that data packets can transmit from and to node.
3. There is no self loop in the network means node is not directly connected to itself, means that the self loop has no meaning.
4. Nodes are finite in the network.

$g \subseteq ND \times ND$ $g = g^{-1}$ $id(ND) \cap g = \varnothing$ $finite(ND)$	$closure \in (ND \leftrightarrow ND) \rightarrow (ND \leftrightarrow ND)$ $\forall r \cdot r \subseteq closure(r)$ $\forall r \cdot closure(r); r \subseteq closure(r)$ $\forall r, s \cdot r \subseteq s \wedge s; r \subseteq s \Rightarrow closure(r) \subseteq s$

The above item-2 represents the symmetric graph of the network. The symmetry of the graph is the representation of the undirected graph by pairs of nodes. The item-3 specified the non-reflexivity of the graph and item-4 specified the finite number of moving nodes in the network. There is one more important property of the graph is connectivity. This property is defined by not connectivity but transitive closure.The fixed point theorem states that when we consider a relation over a set, it can be represented by existing path from

one node to other node and define the transitive closure of the set is a relation. The fixed point theorem which can be used as an induction rule and it also ensure that the nodes are connected with help of many elementary pairs of nodes.
The following axioms $(axm1, axm2)$ represent that the variable $(source)$ and $(target)$ representing a function mapping data packet (Msg) to set of nodes (ND). It means that each data packet has a source and destination node.

$$
\begin{array}{l}
axm1 : source \in Msg \rightarrow ND \\
axm2 : target \in Msg \rightarrow ND
\end{array}
$$

In our abstract model we have introduced the three new variables ($sent$, got and $lost$). The variable $(sent)$ is the set of sent data packets from any source node to the destination node in the network. The second variable (got) is set of successfully received data packets through any destination node in the network. The third variable $(lost)$ is set of lost data packets due to failure of transmission of the data packet in ad-hoc network.The invariant $(inv4)$ represent $(ALinks)$ relation between set of nodes $(ND \leftrightarrow ND)$. $ALinks$ is set of active links in the dynamic changing ad-hoc network.It is always keeping upto-date information of all adding and removing links in the network.

$$
\begin{array}{l}
inv1 : sent \subseteq Msg \\
inv2 : got \subseteq Msg \\
inv3 : lost \subseteq Msg \\
inv4 : ALinks \in ND \leftrightarrow ND
\end{array}
$$

The safty properties of our model is to maintain data packets information in the changing network, means if data packet is successfully received by any target node or losting of data packet in the mid of communication channel.The following invariants represent the safety properties of the ad-hoc network protocol.The invariant $(inv5)$ states that all the data packets received by (got) and $(lost)$ variable is subset of $(sent)$ variable, means all sent data packets sended by source node either successfully received by the destination node or lost due to transmission failure. The next invariant $(inv6)$ states that there is no common data packet in the ad-hoc network which received by the (got) and $(lost)$ variables, means successfully transmitted data packets from source node to destination node represented by the (got) variable and unsuccessfully transmitted data packets from source node to destination node represented by the $(lost)$ variable.

$$
\begin{array}{l}
inv5 : got \cup lost \subseteq sent \\
inv6 : got \cap lost = \varnothing
\end{array}
$$

In the abstract model the data packet will transfer between two nodes in a single atomic step. This provides for a clear and simple abstraction of the essence of the protocol. However, in the real ad-hoc network wireless protocol the sender determine the complete sequence of nodes through which forward the data packet. The sender explicitly lists this route in the packets header, identifying each forwarding by the address of the next node to which, transmit the data packet on its way to the destination node. The protocol presented here is explicitly designed for use in the wireless environment of an ad-hoc network. There are no periodic router advertisements in the protocol. Instead, when a node needs a route to another node, it dynamically determines one based on local routing table or route cached information and on the results of a route discovery protocol. We have introduced the new events and variables in forthcoming models as refinement in the incremental development of the the ad-hoc network system.In abstract specification of the ad-hoc network wireless protocol includes events modeling, atomic transfer of data packets between moving nodes, successfully receiving of data packets by destination node, losing of data packets and routes changes due to movement in nodes.There are five significant events in our abstract model as follows:
The event $(sending)$ represents the sending of data packet $(data_msg)$ from source node (s) to destination node (t).The guards ($grd1$ and $grd2$) state that the data packet is not member of set $(sent)$, which is sending from source node (s) to destination node (t). The other guards of $(sending)$ event state that source node (s) and destination node (t) are different nodes and guard $(grd6)$ states that source node and destination node connected with valid route in ad-hoc network.

EVENT sending
ANY
s,t,data_msg
WHERE
grd1 : $data_msg \in Msg$
grd2 : $data_msg \notin sent$
grd3 : $s \in ND \land t \in ND \land s \neq t$
grd4 : $source(data_msg) = s$
grd5 : $target(data_msg) = t$
grd6 : $s \mapsto t \in closure(ALinks)$
THEN
act1 : $sent := sent \cup \{data_msg\}$
END

The event $(receiving)$ represents the successful receiving of data packet $(data_msg)$ by the destination node (t). The guards of $(receiving)$ event state that after sending the data packet $(data_msg)$ from source node to destination node, should not be received by the either (got) or $(lost)$ variables and source node of the data packet $(data_msg)$ should be source node (s) and destination node (t).

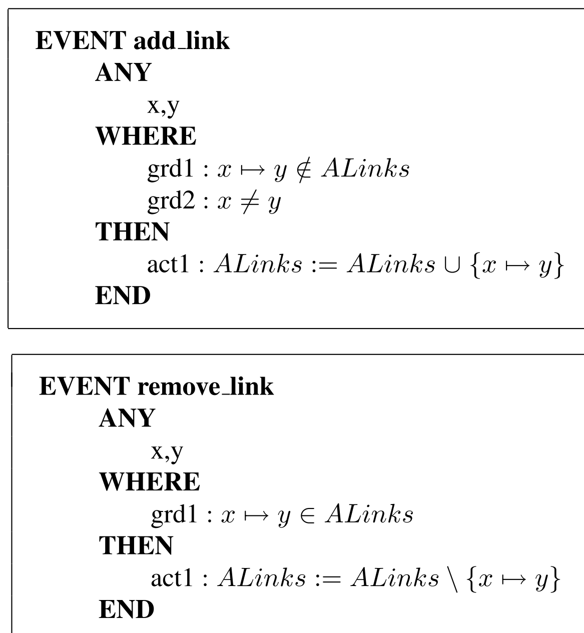
EVENT receiving
ANY
s,t,data_msg
WHERE
grd1 : $data_msg \in sent \setminus (got \cup lost)$
grd2 : $source(data_msg) = s \land target(data_msg) = t$
THEN
act1 : $got := got \cup \{data_msg\}$
END

The event $(losing)$ represents loss of data packets due to network failure or suddenly powered off of any node or moving of node to new location and disconnected from the whole network. The guard of $(losing)$ event state that after sending the data packet $(data_msg)$ from source node to destination node, should not be received by the either (got) or $(lost)$ variables and guard $(grd3)$ states that there is link failure from source node (s) to destination node (t) when data packet $(data_msg)$ on any intermediate node from source node (s) to destination node (t) or in other word we can say node (s) and node (t) are disconnected in the network.

EVENT losing
ANY
s,t,data_msg
WHERE
grd1 : $data_msg \in sent \setminus (got \cup lost)$
grd2 : $source(data_msg) = s \land target(data_msg) = t$
grd3 : $s \mapsto t \notin closure(ALinks)$
THEN
act1 : $lost := lost \cup \{data_msg\}$
END

We know that in wireless adhoc network, no any fixed infrastructure and every node in the network work as router and all nodes are moving from one place to other place without giving any information, so network link infromation is always changing.For modeling this dynamic behaviour the system we proposed the two events add_link and $remove_link$.This event always keeping the uptodate information of the adhoc network.The (add_link) event indicates that adding of new link as elementary path from node (x) to node (y) in ad-hoc network which is not available in current network $(ALinks)$. Similarly the $(remove_link)$

event removing of link from node (x) to node (y) from ad-hoc network which is already connected to current network $(ALinks)$.



EVENT add_link
ANY
x,y
WHERE
grd1 : $x \mapsto y \notin ALinks$
grd2 : $x \neq y$
THEN
act1 : $ALinks := ALinks \cup \{x \mapsto y\}$
END

EVENT remove_link
ANY
x,y
WHERE
grd1 : $x \mapsto y \in ALinks$
THEN
act1 : $ALinks := ALinks \setminus \{x \mapsto y\}$
END

Now our main goal is to discover hidden details between abstract specifications to the concrete machine of the ad-hoc wireless network protocol with stepwise refinement.

6.3.1 First Refinement

In the abstract model we have presented that data packet has been transferred in atomic step from source node to the destination node. But in real ad-hoc network wireless protocol the data packet is transferred hop by hop from source node to destination node.So our goal is to model the store and forward architecture, where all nodes are not directly connected, and data packet must pass through a number of intermediate nodes before reaching to destination node. In the first refinement step, we introduce data structure more closely to store and forward architecture, and introduce the internal action for passing data packets between these data structure.The basic block diagram of first refinement of ad-hoc network protocol is shown in Figure-2. The block diagram represents the store and forward architecture of ad-hoc network. In this refinement we have introduced the variable $(gstore)$, which shown in figure as box $(Store)$ at all distributed nodes in ad-hoc network and all of them can exchange the data packets.

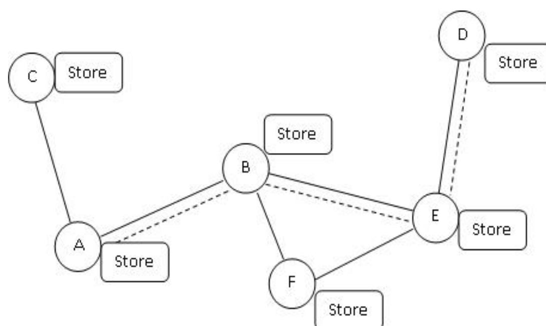


Figure 6.2: Ad-hoc Networks of First refinement

Here we introduced the new variable $(gstore)$ relation between the set of nodes (ND) and set of data packets (Msg). The invariant $(inv2)$ of this refinement states that after sending the data packet $(data_msg)$ from source node to destination node, the data packets stored at intermediate node variable $(gstore)$ before received by target node. The next invariant $(inv3)$ states that total number of sending data packets

of set $(sent)$ should be equal to received data packets of sets (got), $(lost)$ and $(gstore)$ variables. The invariant $(inv4)$ states that all data packets which stored on variable $(gstore)$ should be member of $(sent)$ variable.The invariant $(inv5)$ states that if data packet $(data_msg)$ is not the member of set $(sent)$ then it will be never the member of sets $(got$,$lost$ and $gstore)$. The last invariant of this refinement states that if two nodes maps the same data packets then it will be same node in the $(gstore)$ variable. For preserving the store and forward architecture in ad hoc network, there are all strong invariants which explain above are giving in following figure:-

$inv1 : gstore \in ND \leftrightarrow Msg$
$inv2 : \forall i \cdot i \in ND \land i \in dom(gstore) \Rightarrow (got \cup lost) \cap gstore[\{i\}] = \varnothing$
$inv3 : ran(gstore) \cup (got \cup lost) = sent$
$inv4 : \forall i \cdot i \in ND \Rightarrow gstore[\{i\}] \subseteq sent$
$inv5 : \forall m \cdot m \in Msg \land m \notin sent$
$\Rightarrow$
$m \notin got \quad \land$
$m \notin lost \quad \land$
$(\forall i \cdot i \in ND \Rightarrow i \mapsto m \notin gstore)$
$inv6 : \forall m, i, j \cdot i \mapsto m \in gstore \land j \mapsto m \in gstore \Rightarrow i = j$

A new event $(forward)$ introduce in this refinement, which used to transfer the data packets between neighbouring nodes in the route and it is supposed to be there is direct link between node (x) and node (y).The guard $(grd1)$ of this refinement states that data packet $(data_msg)$ should be member of only set $(sent)$. The guard $(grd2)$ states that there is direct link between two nodes and link is member of current active network $(ALinks)$. The guards $(grd3, grd4)$ of this event state that node (t) is the destination node of data packet and node (x) is not the final destination node. The last two guards $(grd5, grd6)$ state that node (x) map to data packet $(data_msg)$ should be member of $(gstore)$ variable, means data packet $(data_msg)$ is stored on node (x) in ad-hoc network and data packet is not member of $(gstore)$ at node (y). If event will satisfy all the guards, then intermediate node (x) will transfer the data packet $(data_msg)$ to next neighbouring node (y) of the route.

EVENT forward
ANY
t,x,y,data_msg
WHERE
grd1 : $data_msg \in sent \setminus (got \cup lost)$
grd2 : $x \mapsto y \in ALinks$
grd3 : $target(data_msg) = t$
grd4 : $x \neq target(data_msg)$
grd5 : $x \mapsto data_msg \in gstore$
grd6 : $y \mapsto data_msg \notin gstore$
THEN
act1 : $gstore := (gstore \setminus \{x \mapsto data_msg\}) \cup \{y \mapsto data_msg\}$
END

In the first refinement of this model we introduced the new variable $(gstore)$. In the refinement of $(sending)$ event we have added a new guard and an action. The new guard represents that $s \mapsto data_msg$ should not be member of $(gstore)$ variable and the new action represents that $s \mapsto data_msg$ should be added to $(gstore)$ variable.

EVENT sending
ANY
s,t,data_msg
WHERE
$\oplus$ $grd7 : s \mapsto data_msg \notin gstore$
THEN
$\oplus$ $act2 : gstore := gstore \cup \{s \mapsto data_msg\}$
END

In the refinement of event ($receiving$), we have added a new guard and an action. The new guard represents that $t \mapsto data_msg$ should be member of ($gstore$) variable and the new action represents that $t \mapsto data_msg$ should be removed from the $gstore$ variable at the destination node.

EVENT receiving
ANY
s,t,data_msg
WHERE
$\oplus$ $grd4 : t \mapsto data_msg \in gstore$
THEN
$\oplus$ $act2 : gstore := gstore \setminus \{t \mapsto data_msg\}$
END

In the refinement of event ($losing$), a new added guard represents that data packet ($data_msg$) is member of ($gstore$) variable and a new added action represents that $t \mapsto data_msg$ should be removed from the ($gstore$) variable from the intermediate node (x).

EVENT losing
ANY
s,t,data_msg
WHERE
$\oplus$ $grd4 : x \mapsto data_msg \in gstore$
THEN
$\oplus$ $act2 : gstore := gstore \setminus \{x \mapsto data_msg\}$
END

6.3.2 Second Refinement

The basic requirements of the ad-hoc network system are basic communication protocol and route discovery protocol.So we have introduced the two phases of ad-hoc network model.The basic communication protocol is the first phase and route discovery protocol is second phase of our ad-hoc network model. We have introduced the abstract specification in machine M0 and stepwise refinement in machine M1 and machine M2 of basic communication protocol. In this refinement we added new event ($update_routing_table$) for taking the new changes in routing table and this event is triggered whenever source node wants to transmit the data packets to any destination node and there is not available a single route in local routing table of source node.The route discovery protocol is another phase of our model and abstract specification of this model is event ($update_routing_table$) in the second refinement of the basic communication protocol or first phase of the ad-hoc network system.In the development of ad-hoc network system we proposed the following idea, how to model the integration of sub system of same domain and each subsystem of the system has different functionality and main system of the domain is dependent on subsystem for some activities.

The abstract specification of route discovery protocol discover the route and update the routing table of all other nodes in atomic step. In the next refinement of ad-hoc network we introduced the all hidden details of route discovery protocol.Route discovery protocol is a subsystem in ad-hoc network wireless protocol.Route Discovery allows any node in the ad-hoc network to dynamically discover a route to any other node in the

ad-hoc network, whether directly reachable within wireless transmission range or reachable through one or more intermediate network hops through other nodes. In case of changing in network topology or broken route the route discovery protocols used to discover the new route from source node to destination node.For example, when source node s is communicating with destination node t, the node s sends data packets to node t along with the selected route. During their communication, if the node s gets to know that the communication route is broken, the node s does not need to rediscover a new route immediately because node s might have detected several routes in the previous discovery. It can then choose another available route and replace the broken one. Until all the routes are not reachable to the destination node, the system will start route discovery again.

In the second refinement of ad-hoc network , the block diagram in Figure-3 represents the local routing table resides at every node and whenever any source node wants to transfer the data packet to any other node then it select the route from the local routing table.In this refinement the source node and all connected nodes are updating their routing table in one shot.

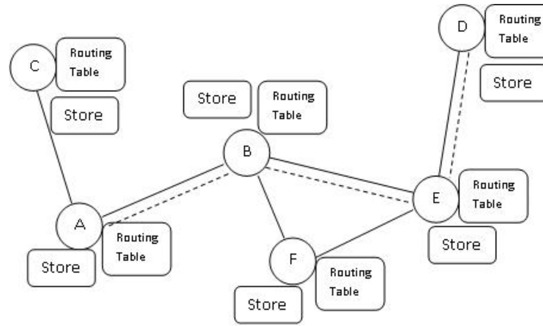


Figure 6.3: Ad-hoc Networks of Fifth refinement

This is a relatively complex and important refinement in which introduce a variable ($alinks$) representing a total function mapping set of nodes (ND) to relation between set of nodes ($ND \leftrightarrow ND$). The variable ($alinks$) contains only set of links information between node to node.The following invariant represents the type of ($alinks$).

$$inv1 : alinks \in ND \rightarrow (ND \leftrightarrow ND)$$

In this refinement, we introduce a new event ($update_routing_table$) for updating the routing table ($alinks$) which reside at each node (ND). All routing tables are updated through event ($update_routing_table$) when route is not available in routing table at the time of sending the data packets to any destination node by using the route discovery protocol.The guard ($grd1$) states that node (s) and node (t) are member of set of nodes (ND). The guard ($grd2$) represents the typr of variable ($routeSet$) and guard ($grd3$) states that there is no link between source node (s) to destination node (t) in routing table set ($alinks$). The variable ($alinks$) always keep some stale links information due to continue changing the node location in ad-hoc network.The guard ($grd4$) states that E is subset of connected nodes between source node to any node of the ad-hoc network in current active link set ($ALinks$). The final guard ($grd5$) of this event find the set of links from set of nodes (E) in the current active link set ($ALinks$).The action of this event states that the set of nodes (E) are updating their routing table.

EVENT update_routing_table
ANY
s,t,E,routeSet
WHERE
grd1 : $s \in ND \wedge t \in ND$
grd2 : $routeSet \in ND \leftrightarrow ND$
grd3 : $s \mapsto t \notin closure(alinks(s))$
grd4 : $E \subseteq \{x|x \in ND \wedge s \mapsto x \in closure(ALinks)\}$
grd5 : $routeSet \subseteq closure(E \lhd ALinks)$
THEN
act1 : $alinks := alinks \Leftarrow (\lambda n \cdot n \in E|alinks(n) \cup routeSet)$
END

In this refinement, the event $(sending)$ keeps all the invariants which explained already of this refinement and adding a new guard in this event. The new guard state that there is connection between source node (s) to destination node (t) or there is valid route from source node (s) to destination node (t) then source node will transfer the data packets.

EVENT sending
ANY
s,t,data_msg
WHERE
$\oplus$ $grd8 : s \mapsto t \in closure(alinks(s))$
THEN
...
END

There is no changing in the refinement of event $(receiving)$. But a new guard and is adding and an old guard is removing from event $(losing)$, which shown in following figure. The new added guard states that in the local routing table of node (x) has not proper path from source node (s) to destination node (t) or node (s) and node (t) are disconnected.

EVENT losing
ANY
s,t,x,data_msg
WHERE
$\ominus$ $grd4 : s \mapsto t \notin closure(ALinks)$
$\oplus$ $grd4 : s \mapsto t \notin closure(alinks(x))$
THEN
...
END

In the refinement of $(forward)$ event a new guard is adding and it states that from any intermediate node (y) to destination node (t) is connected, then the node (x) can forward the data packets to next neighbouring node (y) of the network.

EVENT forward
ANY
x,y,data_msg
WHERE
$\oplus$ $grd7 : y \mapsto t \in closure(alinks(x))$
THEN
...
END

6.3.3 Third Refinement : Route Discovery Protocol

We have explained already that route discovery protocol is the second necessary phase of ad-hoc network protocol for finding the new available route in active network and we have described the abstract specification of the route discovery protocol in last refinement as an event ($update_routing_table$).This event can be triggered by any two event ($sending$) and ($forwarding$), in case of network changing and there is no more links or path are available for sending the data packets to destination node.In the third and next refinements we will describe only the modeling of route discovery protocol.Route Discovery protocol allows any node in the ad-hoc network to dynamically discover a route to any other node in the ad-hoc network, whether directly reachable within wireless transmission range or reachable through one or more intermediate network hops through other nodes. In case of changing in network topology or broken route the route discovery protocols used to discover the new route from source node to destination node.
The abstract specification of route discovery protocol discover the route in atomic step. In this refinement we have introduced only two events ($broadcast_rrq$) and ($received_rrq$) for discovering the new routes. The event ($broadcast_rrq$) send the route request packet, which may be received by those nodes within wireless transmission range of it.
The route request packet identifies the node, referred to as the destination node of the route discovery, for which route is requested.If the route discovery is successful and the source node receives a route reply packet listing a sequence of network hops through which it may reach the destination node using event ($received_rrq$). We defined the constant (rrq) as a set of route request packets, which used to broadcast for discovering the new route from the source node. We introduced the two constant ($source_rrp$) and ($target_rrp$) represent the total function maps a set of route request packets (rrq) to set of nodes (ND) for defining the source node and target node with each route request packet respectively.The other constant (rrp) introduced here for representing the set of route reply packets, it is returned by any destination node when route discovery process has been completed using route request packets. The constant ($source_rrp$) represents total function maps a set of route reply packets (rrp) to set of nodes (ND) for initializing the source node for each route reply packets.The last constant ($seqNo$) represents the total function maps a set of route request packets (rrp) to set of Natural numbers.It is sequence no. which is different for every route request packet

$$
\begin{aligned}
&axm1 : source_rrq \in rrq \rightarrow ND\\
&axm2 : target_rrq \in rrq \rightarrow ND\\
&axm3 : source_rrp \in rrp \rightarrow ND\\
&axm4 : seqNo \in rrq \rightarrow \mathbb{N}_1
\end{aligned}
$$

in this refinement we introduced the two invariants.The invariant ($inv1$) states that the variable ($bcast_rrq$) is subset of (rrq) and whenever any node will broadcast the route request packet then the new route request packet will be member of variable ($bcast_rrq$).Similarly other invariant ($inv2$) states that the variable ($network_rrp$) is subset of (rrp) and any route reply packet will be member of this subset when the target node will return the route reply packet to the source node.

$$
\begin{aligned}
&inv1 : bcast_rrq \subseteq rrq\\
&inv2 : network_rrp \subseteq rrp
\end{aligned}
$$

We are not describing here in this refinement of other events of first phase model (basic communication protocol) due to no changing in subsequent refinements of events.Here we will introduces only the new events of second phase of this model or route discovery protocol.
In this refinement of Route discovery protocol, there are two events $broadcast_rrq$ and $received_rrq$. The event $broadcast_rrq$ broadcast the route request packet to other nodes for discovering the route for any destination node. The guard ($grd1$) states that node (s) and node (t) is member of set (ND). The guard ($grd2$) represents that new route requested packet (rrq_pkt) should not be member of subset broadcasted route request packet ($bcast_rrq$). The guard ($grd3$) states that there is not available path from source node (s) to destination node (t).This guard is essential condition for triggering this event because at the time of sending or forwarding the packet, each node always check the path from local routing table for destination node.If path will not available then the node will broadcast the route request packet for discovering the new

path in active current network.The guards $(grd4, grd5)$ state that each route request packet has source and destination node. The action $(act1)$ of this event states that if all the guards will satisfy then it will broadcast the route request packet in network for discovery the new route.

EVENT broadcast_rrq
 ANY
  s, t, rrq_pkt
 WHERE
  grd1 : $s \in ND \wedge t \in ND$
  grd2 : $rrq_pkt \notin bcast_rrq$
  grd3 : $s \mapsto t \notin closure(alinks(s))$
  grd4 : $source_rrq(rrq_pkt) = s$
  grd5 : $target_rrq(rrq_pkt) = t$
 THEN
  act1 : $bcast_rrq := bcast_rrq \cup \{rrq_pkt\}$
 END

The other new event $(received_rrq)$ of this refinement send the route reply packet to the source node with discovered route information in the network. The guard $(grd1)$ states that node (s) and node (t) is member of set (ND). The guard $(grd2)$ represents that new route requested packet (rrq_pkt) should be member of subset broadcasted route request packet $(bcast_rrq)$. The guard $(grd3, grd4)$ states that target node of the route request packet is (t) and source node of the route request packet is not equal to (t).This condition have very strong properties that if any node received the route request packet, it should be only and only target node.The last guard $(grd5)$ states that returning route reply packet should not be member of subset $(network_rrp)$.The action $(act1)$ of this event states that if all the guards will satisfy then it will send the route reply packet to the source node and action $(act2)$ states that the route request packet will be removed by target node from the network after successfully discovering of route.

EVENT received_rrq
 ANY
  s, t, rrq_pkt, rrp_pkt
 WHERE
  grd1 : $s \in ND \wedge t \in ND$
  grd2 : $rrq_pkt \notin bcast_rrq$
  grd3 : $target_rrq(rrq_pkt) = t$
  grd4 : $source_rrq(rrq_pkt) \neq t$
  grd5 : $rrp_pkt \notin network_rrp$
 THEN
  act1 : $network_rrp := network_rrp \cup \{rrp_pkt\}$
  act2 : $bcast_rrq := bcast_rrq \setminus \{rrq_pkt\}$
 END

6.3.4 Fourth Refinement : Route Discovery Protocol

In the last refinement of route discovery protocol we introduced only two new events but in real route discovery protocol the complete route has been discovered using many several process or events.In this refinement we have tried to discover more and more information about route discovery protocol. In addition to the address of the original initiator of the request and the target of the request, each route request packet contains a route record, in which is accumulated a record of the sequence of hops taken by the route request packet as it is propagated through the ad-hoc network during this route discovery.Each route request packet also contains a unique request id known as sequence number has been introduced in next refinement.In this refinement of the system we have introduced the four more invariants and a new event $(forward_rrq)$. A new variable $(route_record_rrq)$ representing a total function mapping set of route request packets (rrq) to relation between set of nodes $(ND \leftrightarrow ND)$.This variable used to store the link information at the time of propagation of route request packets from one node to other node.If the route request receiver node is not the

target node then it add the link information to the route record of route request packet and again broadcast it. Similarly other new variable $(route_record_rrp)$ representing a total function mapping set of route reply packets (rrp) to relation between set of nodes $(ND \leftrightarrow ND)$.This variable also used to store the link information which is collected by the route request packet, when destination node returned the route reply packet to the source node. There are two more invariants $(inv3, inv4)$ introduced which shown in below figure.The invariants $(inv3, inv4)$ are required to prove some proof obligation.The following invariants represent that sequence of accumulated node information and route record information is subset of all connected nodes to source node and subset of links of connected links from source node to other nodes respectively. We can't check this properties with current active links set $(ALinks)$, because this is too strong property and it is possible that after the completion of route discovery process any search link can be disappear.So we assumed that links are valid for a while and nodes are moving with moderate rate and not any link will be disappear during route discovery process.In the real network protocol this process is happenig so fast that network structure look like constant, so our assumptions are correct and proved by rodin tool.

$inv1 : route_record_rrq \in rrq \rightarrow (ND \leftrightarrow ND)$
$inv2 : route_record_rrp \in rrp \rightarrow (ND \leftrightarrow ND)$
$inv3 : \forall rp, al, s \cdot s \in ND \wedge rp \in rrp \wedge al \subseteq ND \times ND \wedge al \in dom(closure)$
$\Rightarrow$
$dom(route_record_rrp(rp)) \subseteq \{x \cdot s \mapsto x \in closure(al) | x\}$
$int4 : \forall al, E, rp \cdot E \subseteq ND \wedge rp \in rrp \wedge al \subseteq ND \times ND \wedge al \in dom(closure)$
$\Rightarrow$
$route_record_rrp(rp) \subseteq closure(E \lhd al)$

The new event $(forward_broadcast)$ introduced in this refinement for broadcasting the route request packet to the neighboring node when any node is not the target node for route discovery process.The guard $(grd1)$ states that node (x) and node (y) is member of set nodes (ND).The second guard showing that route request packet (rrq_pkt) should be member of subset $bcast_rrq$).The third guard $(grd3)$ states that there is proper connection between neighboring nodes(x, y).And last two guards $(grd4, grd5)$ state that node (y) is not source node and as well as destination node of the route request packet.As the actions of this event $(act1, act2)$ state that when all guard will satisfy then the first action will add new link information $(x \mapsto y)$ to the route request packet and again broadcast it continue for route discovery process.This process will be repeated many time, until the destination node will not receive the route request packet.

EVENT forward_broadcast
ANY
x, y, rrq_pkt
WHERE
grd1 : $s \in ND \wedge t \in ND$
grd2 : $rrq_pkt \in bcast_rrq$
grd3 : $x \mapsto y \in alinks(x)$
grd4 : $source_rrq(rrq_pkt) \neq y$
grd5 : $target_rrq(rrq_pkt) \neq y$
THEN
act1 : $route_record_rrq(rrq_pkt) := route_record_rrq(rrq_pkt) \cup \{x \mapsto y\}$
act2 : $bcast_rrq := bcast_rrq \cup \{rrq_pkt\}$
END

In this refinement, we are adding two new guards and removing two old guards from the event $(update_routing_table)$.New two added guards are more specific then the previous refinement old guards.The changed guards are shown in the following figure. We have added new invariants $(inv3, inv4)$ for proving the proof obligation in this refinement which already explain in the beginning of the this refinement.

EVENT update_routing_table
ANY
s,t,E,routeSet,rrp_pkt
WHERE
$\ominus$ $grd3 : E \subseteq \{x|x \in ND \wedge s \mapsto x \in closure(ALinks)\}$
$\ominus$ $grd4 : routeSet \subseteq closure(E \lhd ALinks)$
$\oplus$ $grd3 : E = dom(route_record_rrp(rrp_pkt))$
$\oplus$ $grd4 : routeSet = route_record_rrp(rrp_pkt)$
THEN
...
END

There is no changing in event $(broadcast_rrq$ of this refinement while in the event $(received_rrq)$ we are adding two extra guards which shown in the following figure.These two guards state that in the discovered route by route request packet has proper path from source node (s) to the destination node (t) and route record of route request packet is equal to the route record of route reply packet, and route reply packet will send to the source node for updating the route information.

EVENT received_rrq
ANY
s, t, rrq_pkt, rrp_pkt
WHERE
$\oplus$ $grd6 : s \mapsto t \in closure(route_record_rrq(rrq_pkt))$
$\oplus$ $grd7 : route_record_rrp(rrp_pkt) = route_record_rrq(rrq_pkt)$
THEN
...
END

6.3.5 Fifth Refinement : Route Discovery Protocol

In this refinement of the route discovery protocol we introduced the new variable $(store_rrq)$, which used to store the route request packet at every node. Each route request packet also contains a unique request id, set by the initiator from a locally-maintained sequence number. In order to detect duplicate route requests received, each host in the ad hoc network maintains a list of route request packet that it has recently received on any route request.

The route request thus propagates through the ad hoc network until it reaches the target host, which then replies to the initiator. The original route request packet is received only by those hosts within wireless transmission range of the initiating host, and each of these hosts propagates the request if it is not the target and if the request does not appear to this host to be redundant. Discarding the request because the host's address is already listed in the route record guarantees that no single copy of the request can propagate around a loop. Also discarding the request when the host has recently seen one with the same route request packet removes later copies of the request that arrive at this node by a different route.

In order to return the route reply packet to the initiator of the route discovery, the target host must have a route to the initiator. If the target has an entry for this destination in its route cache, then it may send the route reply packet using this route in the same way as is used in sending any other packet. Otherwise, the target may reverse the route in the route record from the route request packet, and use this route to send the route reply packet. This, however, requires the wireless network communication between each of these pairs of hosts to work equally well in both directions, which may not be true in some environments.

The new variable $store_rrq$ representing a total function mapping set of nodes (ND) to power set of route request message (rrq). The following invariant represents the type of $(store_rrq)$.

$$inv1 : store_rrq \in ND \rightarrow \mathbb{P}(rrq)$$

In this refinement we introduced the new event ($forward_broadcast_skip$).This event will be triggered when any node already received the route request packet and store in local route request packet store variable ($store_rrq$).The guards ($grd1 - grd5$) already explain in the event ($forward_broadcast$) but two new guards ($grd6, grd7$) are adding in this new events.The guard ($grd6$) states that route request packet (rrq_pkt) is already stored in the local route request packet store variable ($store_rrq$) of intermediate node (y).The guard ($grd7$) states that the sequence no. or request id of the route request packet in set of route request packet at any intermediate node (y).

EVENT forward_broadcast_skip
  ANY
    x, y, rrq_pkt
  WHERE
    grd1 : $s \in ND \wedge t \in ND$
    grd2 : $rrq_pkt \in bcast_rrq$
    grd3 : $x \mapsto y \in alinks(x)$
    grd4 : $source_rrq(rrq_pkt) \neq y$
    grd5 : $target_rrq(rrq_pkt) \neq y$
    grd6 : $rrq_pkt \in store_rrq(y)$
    grd7 : $seqNo(rrq_pkt) \in \{p \cdot p \in store_rrq(y) | seqNo(p)\}$
  THEN
    skip
  END

There is no changing in the events of ($update_routing_table$) and ($received_rrq$) of this refinement. In the event ($broadcast_rrq$) we have added only one new action which shown in the following figure. The new added action ($act2$) in the refinement of this event states that when any route request initiator node broadcast the route request packet then it will also store the route request packet to local store variable ($store_rrq$).

EVENT broadcast_rrq
  ANY
    s, t, rrq_pkt
  WHERE
    ...
  THEN
    $\oplus$ $act2 : store_rrq(s) := store_rrq(s) \cup \{rrq_pkt\}$
  END

In the event (($forward_broadcast$) of this refinement, we are adding two new guards and one extra action. The guard ($grd6$) states that route request packet (rrq_pkt) is not stored in the local route request packet store variable ($store_rrq$) of node (y) and the guard ($grd7$) states that the sequence no. or request id of the route request packet not in set of route request packet at node (y).The new added action ($act3$) states that when any route request receiver node broadcast the route request packet then it will store the route request packet to local store variable ($store_rrq$) of the node.

EVENT forward_broadcast
  ANY
    s, t, rrq_pkt,
  WHERE
    $\oplus$ $grd6 : rrq_pkt \notin store_rrq(y)$
    $\oplus$ $grd7 : seqNo(rrq_pkt) \notin \{p \cdot p \in store_rrq(y) | seqNo(p)\}$
  THEN
    $\oplus$ $act3 : store_rrq(y) := store_rrq(y) \cup \{rrq_pkt\}$
  END

6.4 Conclusion

The stepwise development of adhoc network help us to discover the exact behaviour of basic communication protocol and route discovery protocol.We have developed the whole system in two phases.First phase for basic communication protocol and second phase for route discovery protocol.we have applied the new apporoach for developing the subsystem and integrating them with main system in same domain.Second phase of system is triggerd by an event of first phase. Our contribution are in this paper to model the adhoc wireless network protocol using Event-B and prove it by stepwise refinements and understand the basic notion of adhoc network wireless protocol.Proofs help us to understand the role of graph properties in dynamic environment and correctness of the solution. The refinements gradually introduce the various invariants of the system. No assumption is made on the size of the network but some assuption considering for moving speed of the node. The proof leads us to the discovery of the confirmation event to get the complete correctness, which was not the case of the I/O automata modelling. We have outlined how an incremental refinement approach to the ad-hoc system allowed us to achieve a very high degree of automatic proof.The powerful support provided by the rodin tool was essential to achieving what we believe was a very successful development. Rodin proof was used to generate the hundreds of proof obligations and to discharge those obligations automatically and interactively. Another key role of the tool was in helping us to discover appropriate gluing invariants to prove the refinements. Without this level of automated support, making the changes to the refinement chain that we did make would have been far too tedious. Our approach is the methodology of separation of concerns: first prove the algorithm at an abstract (mathematical) level; then, and only then, gradually introduce the peculiarity of the specific protocol. What is important about our approach is that the fundamental properties we have proved at the beginning, namely the reachability and the uniqueness of a solution, are kept through the refinement process (provided, of course, the required proofs are done). It seems to us that this sort of approach is highly ignored in the literature of protocol developments where, most of the time, things are presented in a flat manner directly at the level of the final protocol itself.

Chapter 7

SmartCards

Sommaire

7.1 Analysing cryptographic protocols **128**
7.2 Pattern for Modelling the Protocols **128**
7.2.1 The Pattern Structure 130
7.2.2 An Example of the Pattern Application 132
7.3 Event-B Models of the Mechanisms **133**
7.3.1 Abstract Model 135
7.3.2 First Refinement 137
7.3.3 Second refinement: attacker's knowledge 137
7.4 Conclusion **139**

Ce chapitre a été rédigé par Nazim Benaissa et Dominique Méry.
We consider the refinement-based process for the development of security protocols, which are very complex systems to specify, to prove and to design. Using existing protocols, we would like to provide proof-based patterns for integrating cryptographic elements in an existing protocol. Our approach is based on incremental development using Event B refinement, which makes proofs easier and which makes the design process faithfull to the structure of the protocol as the designer thinks of it. Communication channels are supposed to be unsafe. Analysing cryptographic protocols requires precise modelling of the attacker's knowledge. The attacker's behaviour conforms to the Dolev-Yao model. In the Dolev-Yao model, the attacker has full control of the communication channel, and the cryptographic primitives are supposed to be perfect. We illustrate the technique on the Shoup-Rubin protocol with smart cards.

7.1 Analysing cryptographic protocols

Cryptographic protocols are complex software systems; they are, therefore, in need of high level modelling tools and concepts. They have also underlying desired properties, which can be expressed logically like secrecy, authenticity. Formal methods are widely used for cryptographic protocols more as a verification tool, not as a design tool. The goal of this paper is to present an attempt to mix the two: design, predominant in software engineering, and formal methods. This leads to the notion of proof-based design allowing a correct-by-construction approach to security protocols.

Our approach is based on incremental development using refinement. The refinement of a formal model allows us to enrich a model in a *step-by-step* approach, and is the foundation of our *correct-by-construction* approach. Refinement is used to make proofs easier but also to make the design process faithfull to the structure of the protocol as the designer thinks of it. Implicit assertions are identified for proving the protocol secure at each refinement step. We use refinement also to introduce smart cards in order to be closer to the protocol implementation and to prove the implementation correct with respect to the protocol security properties.
To be able to prove security properties on a protocol, we must be able to *model the knowledge of the attacker*. A *pet* model of attacker's behaviour is the Dolev-Yao model [41]; this model is an *informal* description of all possible behaviours of the attacker as described by N. Benaissa [33]. Hence, we present a proof-based design to model and prove key protocols using EVENT B [25, 69] as a modelling language. We give a presentation of a proof-based design framewprk and an application of the proof-based design on the Shoup-Rubin protocol [43] with smart cards.
Proving properties on cryptographic protocols such as *secrecy* is known to be undecidable. However, works involving formal methods for the analysis of security protocols have been carried out. Theorem provers or model checkers are usually used for proving properties. For model checking, one famous example is Lowe's approach [47] using the process calculus CSP and the model checker FDR. Lowe discovered the famous bug in Needham-Schroeder's protocol. Model checking is efficient for discovering an attack if there is one, but it can not guarantee that a protocol is reliable with respect to a list of given assumptions on the environment or on the possible attacks. We should be carefull on the question of stating properties of a given protocol and it is clear that the modelling language should be able to state a given property and then to check the property either using model checking or theorem proving. Other works are based on theorem proving: Paulson [50] used an inductive approach to prove safety properties on protocols. He defined protocols as sets of traces and used the theorem prover Isabelle [49]. Other approaches, like Bolignano [35], combine theorem proving and model checking taking general formal method based techniques as a framework. Let us remember that we focus on a correct-by-construction approach and we are not (yet) proposing new cryptographic protocols: we analyse existing protocols and show how they can be composed and decomposed using proved-based design frameworks.

7.2 Pattern for Modelling the Protocols

Our goal is to define a design pattern for modelling cryptographic protocols using EVENT B. The pattern is defined by a proof-based development of EVENT B models which are modelling protocols first with in a very abstract model followed by several refinements. The definition of models is based on the notion of

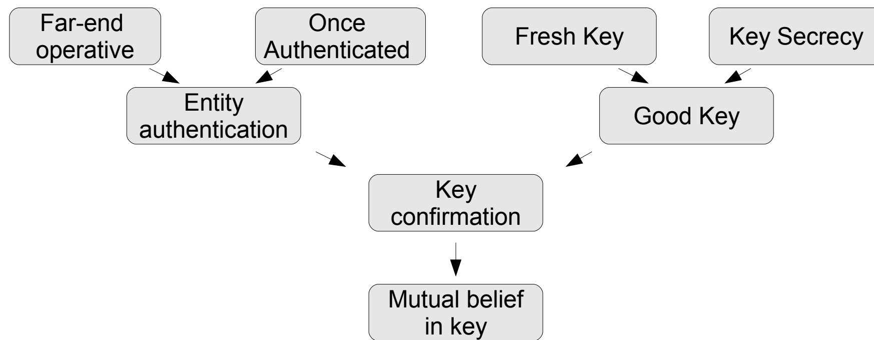


Figure 7.1: Hierarchy of authentication and key establishment goals

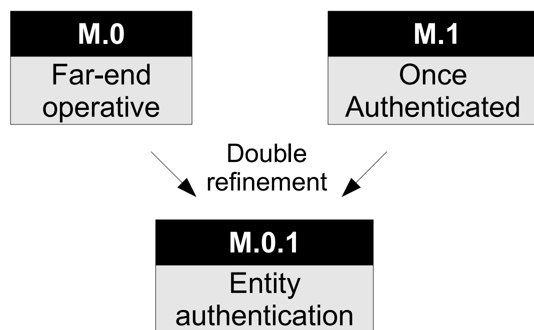


Figure 7.2: Hierarchy of authentication and key establishment goals

transaction. The choice of the details added in each refinement is crucial, this choice is guided by several criteria. First, introducing refinement in general helps to make proofs easier and increases the automatic proofs rate. Second, refinement introduces automation in the design process: in each refinement step, assertions for proving the protocol correct are generated and have to be proved. The hierarchy of the properties to prove corresponds to the hierarchy of the chosen refinements. Boyd and Mathuria [40] give a hierarchy of some important properties that have to be proved on cryptographic protocols in general and properties related with key establishment protocols (Figure 7.1). Far-end operative, peer knowledge, key freshness and key secrecy are generic properties, while the other properties can be obtained by combining these generic ones.

These properties can be divided into two categories, user-oriented goals and key-oriented goals :

- User-oriented goals include far-end operative property and the knowledge of peers among the communication channel. Combining these two generic properties leads to entity authentication.
- Key-oriented goals include key freshness and key secrecy.

Key secrecy also knwon as key authentication means that a shared key is known only by the agents sharing the key and any mutually trusted third party.
The basic idea of our approach is to make the incremental development of the cryptographic protocol faithfull to the structure of the protocol as the engineer thinks of it. What we want to do is to identify all the mechanisms that let a protocol satisfy each safety property, and then combine them by refinement to obtain the final protocol (Figure 7.2).

7.2.1 The Pattern Structure

The properties contained in the figure 7.1 are defined in an abstract way in EVENT B using the notion of abstract transactions that will be introduced in section 7.3. For each property, a set of mechanisms that guarantee this property are available described with an EVENT B models. Let $\mathcal{P}$ the set of properties and $\mathcal{M}$ the set of EVENT B models of mechanisms. For a certain attacker's model, a relation "$\leadsto$" is defined over the sets $\mathcal{P}$ and $\mathcal{M}$:

Definition 1 $\forall p, m.\ p \in \mathcal{P} \wedge m \in \mathcal{M}$*:* $m \leadsto p$ *iff* m *implements the property* p*.*

In section 7.3 we will formally introduce how properties and mechanisms are modelled in EVENT B. We will also see how a mechanism is proved to implement a property.
Let us consider an example of a mechanism using shared key cryptography that implements the authentication property. The mechanism 1 is contained in the *ISO/IEC 9798-2 two-pass unilateral authentication protocol* from the international standard *ISO/IEC 9798 Part 2* [54]:

---

1. $B \rightarrow A : N_b$
2. $A \rightarrow B : \{N_b, B\}_{K_{AB}}$

Mechanism 1

In the mechanism 1, B has knowledge of A as her peer entity, we proved that this mechanism implements the knowledge of peer property in the Dolev-Yao attacker model(see section 7.3). If the nonce N_b is fresh we can also prove that the mechanism implements the Far-end operative property.
We may need to combine several mechanisms to obtain the desired protocol. We combine two mechanisms by performing a double refinement of their corresponding EVENT B models. Proofs of the double refinement have to be performed and a new EVENT B model of a new mechanism is then obtained. We will see in the end of this section how an entire protocol can be obtained by combining several mechanisms.
Combining two mechanisms is not always possible, for example, combining the mechanism 1 with the following mechanism where A has knowledge of B as his peer entity may not be possible.

---

1. $A \rightarrow B : N_a$
2. $B \rightarrow A : \{N_a, B\}_{K_{AB}}$

Mechanism 2

Let us consider a possible composition of the two previous mechanisms:

---

1. $A \rightarrow B : N_a$
2. $B \rightarrow A : \{N_a, B\}_{K_{AB}}, Nb$
3. $A \rightarrow B : \{N_b, B\}_{K_{AB}}$

Mechanism 3

We may think that this new mechanism provides both properties of the composed ones but a possible attack on the new obtained mechanism is as follows:

---

1. $I_A \rightarrow B : N_i$
2. $B \rightarrow I_A : \{N_i, B\}_{K_{AB}}, Nb$
3. $I_B \rightarrow A : N_b$
4. $A \rightarrow I_B : \{N_b, A\}_{K_{AB}}, Na$
5. $I_A \rightarrow B : \{N_b, A\}_{K_{AB}}$

Attack 1

Both mechanisms when considered separatly implements the authentication property but when we combine them, the attacker can use one mechanism to attack the other as shown in the attack 1.
We identified for each mechanisms a set of conditions that guarantee that a combination of this mechanism with others is possible and a set of proof obligations are generated and have to be discharged.
We also introduce a function $\mathcal{R}$ that maps two models $m1$ and $m2$ to the set of possible mechanisms obtained by combinig these two mechanisms in a correct way (proof obligations have been discharged):

Definition 2 $\forall m1, m2, m.\ m1 \in \mathcal{M} \wedge m2 \in \mathcal{M} \wedge m \in \mathcal{M}$*:* $m \in \mathcal{R}(m1, m2)$ *iff* m *is a double refinement of* $m1$ *and* $m2$*.*

The basic idea of our approach is to start with mechanisms that implements generic properties and to combine them by a double refinement process to implement more complex properties as shown in figure 7.1. We have proved the following theorem that we will use to combine mechnisms:

Theorem 1 $\forall m, m1, m2, p1, p2. m \in \mathcal{M} \wedge m1 \in \mathcal{M} \wedge m2 \in \mathcal{M} \wedge p1 \in \mathcal{P} \wedge p2 \in \mathcal{P}$*:*
If $m1 \leadsto p1 \wedge m2 \leadsto p2 \wedge m \in \mathcal{R}(m1, m2)$ *then* $m \leadsto p1 \wedge p2$

When proving a protocol, we will need to prove only the combination proof obligations and not the proofs of different mechanism that are done only once and can be reused for different protocols.
The last definition we need to introduce before prensenting the pattern is the instance of a mechanism. We may need to use several instances of the same mechanism. From an EVENT B point of view two instances of the same mechanism model are simply two models where variables are renamed so these models have exactly the same behaviour and satisfy the same properties. Two instances of the same mechanism are linked with a $\sim$ relation defined as follows:

Definition 3 $\forall m1, m2.\ m1 \in \mathcal{M} \wedge m2 \in \mathcal{M}$*:*
$m1 \sim m2$ *iff* $m1$ *and* $m2$ *are two instances of the same mechanism.*

Theorem 2 $\forall m1, m2, p. m1 \in \mathcal{M} \wedge m2 \in \mathcal{M} \wedge p \in \mathcal{P}$*:*
If $m1 \sim m2 \wedge m2 \leadsto p$ *then* $m1 \leadsto p$

Using our design patterns has two advantages:

- To have an efficient refinement (in term of automatic proofs), we need to make the right choices when choosing abstraction levels and variables during the modelling process. When using the pattern, the designer of the cryptographic protocol will have to decide only how to combine mechanism to obtain the desired protocol since the mechanisms are already modelled in EVENT B.
- Proofs of already proved mechanisms are done once and can be reused with different protocols and proof obligations for combining them are defined.

The pattern is organized in three modules as shown in the table 7.1.

- The LIBRARY module: contains the available properties and the mechanisms that are proved to implement them.
- The COMPOSITION module: contains the structure of the protocol and shows how it was obtained by composing the different mechanisms. This module has three clauses:
  1. The PROPERTIES clause: contains the properties that the protocol should satisfy.
  2. The MECHANISMS clause: contains the instances of the used mechanisms in the protocol.
  3. The THEOREM clause: shows the $\leadsto$ relation of the instances of mechanisms used in the protocol.
- The B_MODELS clause: contains the EVENT B models of the mechanisms.

The pattern can be used for two purposes, designing new protocols and analysing existing ones. By analysing a protocol we mean proving it by identifying which component of the protocol guarantees each property and possibly identify any unnecessary component of the protocol.

Table 7.1: The design pattern for cryptographic protocols

ATTACK_MODEL DY		
LIBRARY	**COMPOSITION**	**B-MODELS**
PROPERTIES	**PROPERTIES**	
$<pi>$	$<pi>$	
...		
$<pj>$	**MECHANISM**	mi_a = B models of the mechanisms
MECHANISM	$mi_a \sim mi$	
$mi \rightsquigarrow pi$		
...	**THEOREM**	
$mj \rightsquigarrow pj$	$mi_a \rightsquigarrow pi$	

1. First the designer chooses the attacker model, this choice is important since a mechanism may implement a property in one attacker model but not in an another one.
2. The designer adds a desired property to the clause ”PROPERTIES”of the ”COMPOSITION” module, the property is chosen among a set of predefined ones contained in the ”LIBRARY” module.
3. A mechanism implementing the added property is chosen among predefined ones in the ”LIBRARY” and an instance of this mechanism is added to the clause ”MECHANISMS” in the ”COMPOSITION” module. The corresponding EVENT B model of the new mechanism is added to the ”MODELS” clause.
4. The designer can perform a composition of two mechanisms. When a double refinement is performed, the resulting model is added to the clause ”MODELS” and a new mechanism resulting from this combination is added to the clause ”MECHANISMS” of the ”COMPOSITION” clause.
5. Proofs obligations of the double refinement are generated and have to be performed, if the double refinement is not possible a different mechanism has to be chosen. Once prrof obligations are discharged the ”THEOREM” clause of the ”COMPOSITION” module is updated.
6. Stop if the desired protocol is obtained or go to step 2.

7.2.2 An Example of the Pattern Application

We applied our design pattern on different cryptographic protocols among which: Blake-Wilson-Menezes key transport protocol [34], the well known Needham-Schroeder public key protocol [48] and the Shoup-Rubin key distribution protocol [43]. We will illustrate our approach in this paper on a simplified version 1 of the well known Kerberos protocol [55], this protocol based on the Needham-Schroeder protocol.

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{K_{AB}, B, N_a\}_{K_{AS}}, \{K_{AB}, A\}_{K_{BS}}$
3. $A \rightarrow B : \{A, T_A\}_{K_{AB}}, \{K_{AB}, A\}_{K_{BS}}$

Protocol 1: *A simplified version of the Kerberos protocol*

The basic protocol involves three parties, the client (A), an application server (B) and an authentication sever (S). A secret long term key K_{AS} is shared between A and S and another key K_{BS} is shared between B and S. This simplified protocol provides the following properties:

	Property	Agent
$p1_A$	Key authentication	A
$p1_B$	Key authentication	B
$p2_A$	Key freshness	A
$p3_B$	far-end operative	B

Three mechanisms are used to guarantee these propeties. The first one (mechanism 4) is a part of the $ISO/IEC11770\ Part2$ [53].

Table 7.2: The design pattern for Kerberos protocol step1

ATTACK_MODEL DY		
LIBRARY	**COMPOSITION**	**B-MODELS**
PROPERTIES $<p1_A>$ $<p1_B>$ $<p2_A>$ $<p3_B>$ ... **MECHANISM** $m4 \leadsto p1_A$ $m4 \leadsto p1_B$ $m5 \leadsto p2_A$ $m6 \leadsto p3_B$ ...	**PROPERTIES** $<p1_A>$ $<p1_B>$ **MECHANISM** $m4_1 \sim m4$ **THEOREM** $m4_1 \leadsto p1_A \wedge p1_B$	$m4_1$ = B models of the mechanism

1. $S \rightarrow A : \{K_{AB}, B\}_{K_{AS}}$
1. $S \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$

Mechanism 4

The second mechanism is similar to mechanism 1 where the server S is involved, but the identity of agent A in the response message is no longer necessary since reflection attack is not possible anymore.

1. $A \rightarrow S : N_a$
2. $S \rightarrow A : \{N_a\}_{K_{AS}}$

Mechanism 5

The last mechanism uses time stamps as follows:

1. $A \rightarrow B : \{A, T_a\}_{K_{AB}}$

Mechanism 6

We present here how the pattern is applied, but due to lack of space we will skip the EVENT B models, in section 7.3 contains the EVENT B model of one mechanism taken as an example. Mechanisms are introduced one by one in the pattern and proofs of double refinement are generated and discharged:

We emphasize that this simplified version of Kerberos protocol does not satisfy key freshness property for agent B. In the full Kerberos protocol uses a mechanism of expiration time in the mssag intended to B to fulfill freshness property and then satisfy the *key confirmation property* for agent B as shown in figure 7.1.

7.3 Event-B Models of the Mechanisms

We present in this section the EVENT B models of the mechanism1 shown before. Our goal is to prove that a mechanism implements a certain property and also to identify the proof obligation of correct composition of this mechanism with others. To prove that a mechanism satisfies a given property we need one abstract model and two refinement steps:

Table 7.3: The design pattern for Kerberos protocol step 2

ATTACK_MODEL DY		
LIBRARY	**COMPOSITION**	**B-MODELS**
PROPERTIES	**PROPERTIES**	
$<p1_A>$	$<p1_A>$	
$<p1_B>$	$<p1_B>$	
$<p2_A>$	$<p2_A>$	
$<p3_B>$	**MECHANISM**	
$\ldots$	$m4_1 \sim m4$	$m4_1 = Bmodels$
MECHANISM	$m5_1 \sim m5$	$m5_1 = Bmodels$
$m4 \rightsquigarrow p1_A$	$m45_1 \in \mathcal{R}(m4_1, m5_1)$	$m45_1 = Bmodels$
$m4 \rightsquigarrow p1_B$	**THEOREM**	
$m5 \rightsquigarrow p2_A$	$m4_1 \rightsquigarrow p1_A \wedge p1_B$	
$m6 \rightsquigarrow p3_B$	$m5_1 \rightsquigarrow p2_A$	
$\ldots$	$m45_1 \rightsquigarrow (p1_A \wedge p1_B) \wedge p2_A$	

Table 7.4: The design pattern for Kerberos protocol step 3

ATTACK_MODEL DY		
LIBRARY	**COMPOSITION**	**B-MODELS**
	PROPERTIES	
	$<p1_A>$	
PROPERTIES	$<p1_B>$	
$<p1_A>$	$<p2_A>$	
$<p1_B>$	$<p3_B>$	
$<p2_A>$	**MECHANISM**	
$<p3_B>$	$m4_1 \sim m4$	$m4_1 = Bmodels$
$\ldots$	$m5_1 \sim m5$	$m5_1 = Bmodels$
MECHANISM	$m45_1 \in \mathcal{R}(m4_1, m5_1)$	$m45_1 = Bmodels$
$m4 \rightsquigarrow p1_A$	$m6_1 \sim m6$	$m6_1 = Bmodels$
$m4 \rightsquigarrow p1_B$	$m456_1 \in \mathcal{R}(m45_1, m6_1)$	$m456_1 = Bmodels$
$m5 \rightsquigarrow p2_A$	**THEOREM**	
$m6 \rightsquigarrow p3_B$	$m4_1 \rightsquigarrow p1_A \wedge p1_B$	
$\ldots$	$m5_1 \rightsquigarrow p2_A$	
	$m45_1 \rightsquigarrow (p1_A \wedge p1_B) \wedge p2_A$	
	$m456_1 \rightsquigarrow ((p1_A \wedge p1_B) \wedge p2_A) \wedge p3_B$	

- The first model is the specification, the desired property is stated in an abstract way using the notion of abstract transaction that will be introduced later in this section. The way a property is expressed in this first abstract model is common to all the mechanisms.
- The second model is the implementation, we exhaustively add all details used by the mechanism to guarantee the desired property stated in the previous model. The RODIN tool will then automatically generate proof obligations by respect to the property stated in the abstract model. We obtain then assertions on the attacker knowledge to preserve the safety properties.
- Third model: we model the behaviour of the attacker. The attacker knowledge is modelled and used to prove the assertions identified in the previous refinement. We introduced attacker's knowledge in a different refinement so we can apply several attacker's behaviours to see if we still can prove safety properties. Note that models corresponding to different attacker behaviours are already defined and are directly applied to the mechanism model.

7.3.1 Abstract Model

The figure 7.3 shows an abstract view of a two-pass unilateral authentication protocol where agent A challenges agent B and waits for the response. After receiving the answer, agent A should be able to authenticate the source of the message. At this abstraction level, we introduce the notion of agent, that is common to all kinds of cryptographic protocols, other notions like nonces, timestamps or cryptographic keys are specific to each kind of protocols and will be introduced in later refinements.
To model the different properties, the pattern is based on the notion of *abstract transactions*. An *abstract transaction* is a session of the protocol performed by one of the agent involved in the protocol run. In some cryptographic protocols, nonces are used to identify each session or protocol run. Intuitively, each transaction of the abstract model will correspond, in this case, to a fresh nonce (or to a timestamp in other protocols) in the concrete model. A transaction has several attributes and, before giving these attributes, we need to introduce the basic sets we will use in our model:

T : is the set of abstract transactions;
$Agent$: is the set of agents;
MSG : is the set of possible messages among agents;
$I \in Agent$: the intruder.

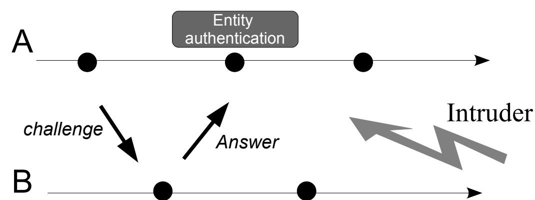


Figure 7.3: Abstract model

Note that for most protocols, even if there is more than one dishonnest agent in the system, it suffices to consider only one attacker that will combine the abilities and knowledge of all the other dishonnest agents. There are several definitions in the literature of entity authentication in cryptographic protocols, Syverson and van Oorschot [52] define entity authentication as: *"A believes B recently replied to a specific challenge."* This property is obtained by combining two generic properties: peer knowledge and far-end operative. In this first model we focus on peer knowledge property. To be able to model it, we introduce variables that model each protocol run or session attributes (one abstract transaction in the abstract model). An abstract

transaction has a source (t_src) that is the agent that initiated the transaction and a *believed* destination (t_bld_dst) that is the believed destination agent. A running transaction is contained in a set $trans$. When a transaction terminates it is added to a set end.

$$trans \subseteq T$$
$$end \subseteq trans$$
$$t_src \in trans \rightarrow Agent$$
$$t_bld_dst \in trans \rightarrow Agent$$

$$answer \subseteq MSG$$
$$answer_src \in answer \rightarrow Agent$$
$$answer_dst \in answer \rightarrow Agent$$

A's challenge and B's answer (figure 7.3) are also modelled using new variables. The answer from the destination agent is transmitted via a channel ($answer$). A message from this channel has a source and a destination ($answer_src$, $answer_dst$). The same variable are added for modelling the challenge.

When A sends a challenge to B, he waits for the answer. Several answers may arrive and A will choose one of them. If the protocol is correct by regard to peer knowledge property, agent A should be able to choose the "appropriate" answer among the ones he received. The source of the chosen message is then considered as the real destination, we store it in a new variable t_dst that has the same type as t_bld_dst. In this case, *to prove peer knowledge, we need to prove that both variables are equal when a transaction terminates.*

Theorem 3 $\forall t \cdot t \in end \wedge t_src(t) \neq I \Rightarrow t_dst(t) = t_bld_dst(t)$

We tried first to prove the property without the condition $t_src(t) \neq I$ but it was not possible to prove it because, since the attacker is sending random messages, he may be mislead by his own messages when he behaves as an honnest agent, we modified the property and we could then prove it.

We emphasise that the goal of this first abstract model is to state the property without explaining how the protocol manage to satisfy it.

Events

At the beginning of a transaction, the agent A sets the value of the variable t_bld_dst to some agent B, adds the transaction t to the set $trans$ and sends the challenge to agent B. Agent B answers by sending the answer to A, the variable $answer_src$ is set to the value B. When an agent A receives an answer from an agent B, he sets the variable t_dst to the value B contained in $answer_src$. Thus, the variable t_dst contains the real transaction destination. The value of this variable is not set in the $Answer$ event, when the agent B sends the message because many agents may answer to agent A's request and the real transaction destination is known only once A receives answer messages and chooses one of them. Depending of the protocol structure, the agent A should know, if the source of the message he received is the trusted destination of the transaction to guarantee the authentication of the protocol. But in this abstract model, we add the *guard 8* that guarantees this property.

```
EVENT End    ANY
        t, A, B, Amsg
    WHERE
        grd1 : t ∈ trans \ end
        grd2 : A ∈ Agent ∧ B ∈ Agent ∧ A ≠ B
        grd3 : t_src(t) = A
        grd4 : Amsg ∈ Answer
        grd6 : msg_src(Amsg) = B
        grd7 : msg_dst(Amsg) = A
        grd8 : msg_src(Amsg) = t_bld_dst(t)
    THEN
        act1 : end := end ∪ {t}
        act2 : t_dst(t) := B
    END
```

We also add in this model the attacker event. In this event the attacker can add a message with randomly chosen attributes to the variables *challenge* and *answer*. Another event modelling the loss of messages is added. Messages are removed from the channels *challenge* and *answer* randomly. This loss can be caused by a malicious attacker action or by an error in the communication channel.

7.3.2 First Refinement

The goal of this first refinement is to understand how the property stated in the previous model is achieved, thus, the corresponding details of the modelled mechanism messages are exhaustively added. Cryptographic keys are introduced in this refinement. For example shared keys are modelled as follows:

KEY Set of all pair keys
$KEY_A \in KEY \rightarrow Agent$ the first owner agent
$KEY_B \in KEY \rightarrow Agent$ the second owner agent

We also need to model nonces at this refinement level. At each protocol run, an agent generates a new nonce and uses it to identify the current session. In this case, each abstract transaction corresponds to a freshly generated nonce. When an agent initiates a new session, he chooses a transaction t that is not in the set $trans$ just like a freshly generated nonce. Instead of adding a new type for nonces, we simply continue to use transactions.
In the step *2* of the mechanism 1 , we need the nonce Na, the identity of the agent and the key used for the encryption.

$answer_Na \in answer \rightarrow T$: the encrypted nonce
$answer_KAB \in answer \rightarrow KEY$
$answer_A \in answer \rightarrow Agent$: the agent identity

In the abstract model, we use the *guard 8* in the **EVENT End** to prove authentication, with this guard agent A could know if the message is authentic or not. In cryptographic protocols, it is not possible to perform such tests but the structure of the answer message itself guarantees authentication. Accordingly, the *guard 8* in the **EVENT End** have to be substituted by a condition over the received answer message content[1].

In general, $accepted_msg$ be the predicate that is true when a message is accepted by the receiving agent. The predicate is directly derived from the protocol itself.
The **EVENT End** of the pattern becomes:

```
EVENT End      ANY
    t, A, B, Amsg
  WHERE
    ⊕ grd8 : accepted_msg(Amsg)
    ⊖ grd8 : answer_src(Amsg) = B
  THEN
    act1 : end := end ∪ {t}
    act2 : t_dst(t) := B
  END
```

7.3.3 Second refinement: attacker's knowledge

To be able to prove properties such as secrecy and authentication on a protocol, we have to be able to model the knowledge of the attacker. To model the knowledge of the attacker, it is necessary to know exactly what the attacker is able to do. One popular model for attacker's behaviour is the Dolev-Yao model. This refinement models all the options the attacker has in this attacking model and can be reused for different protocols. The attacker can then try to obtain nonces and keys from the content of the communication channel. The attacker may also have an initial knowledge, or a knowledge he may acquire by means other than analysing the communication channel content. To model all these options, we use variables that contain the crucial information the attacker can obtain. Because of the typing constraints in the EVENT B, we use one variable for each information type : N_Mem for nonces and K_Mem for each type of keys keys.

$N_Mem \subseteq \mathbb{P}(T)$
$K_Mem \subseteq \mathbb{P}(Key)$

The attacker can also use fragments of encrypted messages contained in the communication channel, we model the set of fragments available to the attacker using a variable $FRAG$. In the case of mechanism 1, the fragment has the following structure:

[1] ⊕ and ⊖ are respectively the added and removed guards compared to the refined event.

$$FRAG \subseteq MSG$$
$$FRAG_Na \in FRAG \rightarrow T$$
$$FRAG_KAB \in FRAG \rightarrow KEY$$
$$FRAG_A \in FRAG \rightarrow Agent$$
$$FRAG_Src \in FRAG \rightarrow Agent$$

We need to answer two issues: What is in the variables *N_Mem* and *K_Mem*? How does the intruder use the knowledge contained in these variables ? The answer of the second issue is immediate, the *event Attack* (for a challenge or an answer) is refined in a way where the attacker uses only transactions or keys that are in his memory and also fragments of encrypted messages contained in the variable $FRAG$.

The *event Attack* is refined into several concrete events that include all the options, we give here one event that models the attacker when using transactions in his memory (without fragment of encrypted messages):

EVENT Attack_Answer ANY
$set_t, set_k, A, Amsg$
WHERE
$\oplus \; grd7 : set_t \subseteq N_Mem$
$\oplus \; grd8 : set_k \subseteq K_Mem$
THEN
$act1 : \oplus MSG_VAR$
END

The answer of the first issue: what is in the attacker memory ? depends from the chosen attacker model. In the Dolev-Yao one, attacker has full control of the communication channels. In our model, we have already added events where messages are lost no matter if it is done by the attacker or not. And we didn't limit the number of messages the attacker can send. To model the fact that an attacker decrypts parts of the message, if he has the appropriate key, we added the following event where the attacker uses keys he knows to decrypt fragments of messages, we do this exhaustively with all *channels*. Events, that models options the attacker has, can be added or removed depending from the chosen attacking model.
To achieve the refinement, we need to prove that the modelled attacker's knowledge preserves the desired property. We use the RODIN tool to generate a carachterization of the attacker's knowledge (variables N_Mem, K_Mem and $FRAG$). In the case of the mechanism 1, we used the RODIN tool and we generated and proved the two following theorems:

Theorem 4

$$\begin{array}{l} \forall K, A, B \cdot \\ \quad K \in K_MEM \wedge \\ \quad K \in dom(KEY_A) \wedge \\ \quad \neg A = I \wedge \\ \quad ((KEY_A(K) = A \wedge KEY_B(K) = B) \vee \\ \quad (KEY_A(K) = B \wedge KEY_B(K) = A)) \\ \quad \Rightarrow \\ \quad B = I \end{array}$$

This first theorem states that to preserve the desired property the attacker should only posses keys he share with another agent. The second theorem is as follows:

Theorem 5

$$\begin{array}{l} \forall A, B, K, frag \cdot \\ \quad frag \in FRAG \wedge \\ \quad FRAG_A(frag) = A \wedge \\ \quad FRAG_Key(frag) = K \wedge \\ \quad ((KEY_A(K) = A \wedge KEY_B(K) = B) \vee \\ \quad (KEY_A(K) = B \wedge KEY_B(K) = A)) \wedge \\ \quad \neg A = I \wedge \\ \quad K \in dom(KEY_A) \\ \quad \Rightarrow \\ \quad B = FRAG_Src(frag) \end{array}$$

Intuitively this theorem states that if a fragment is encrypted with a key owned by two agents and the identity of one of these agent is in the field $FRAG_A$ then the source of this fragment is the other agent. These two theorems are very important for the composition of mechanisms. If this mechanism is composed with another one we need to prove that the fragments of the same type generated by the other mechanism satisfy these theorems. This is how proof obligations of mechanisms composition are generated. When we tried to compose mechanisms 1 and 2 we could not prove these theorems and we could generate the attack 1.

7.4 Conclusion

We have introduced an Event-B-based design pattern for cryptographic protocols and we have applied it on three different protocols. Several properties were proved on these protocols, user-oriented and key-oriented properties. Less than 10% of the proofs of the models were interactive. Patterns facilitate proof process by reusing partially former developments; we have not yet designed new cryptographic protocols and it remains to develop other case studies by applying patterns. Like design patterns, proof-based patterns are based on real cases; they should help the use of refinement and proof techniques; it is then clear that specific tools should be developed and further works should be carried out using refinement for discovering new patterns. As a perspective of our work, we want to model more mechanisms and define a plugin of the RODIN tool that implements this pattern. It is also necessary to address questions on extensions of Dolev-Yao models.

Chapter 8

Self-healing systems

CONTEXT COLOR
SETS
  $color$
CONSTANTS
  $clr1, clr2, clr3$
AXIOMS
  $axm1 color = \{clr1, clr2, clr3\}$
  $axm2 clr1 \neq clr2$
  $axm3 clr1 \neq clr3$
  $axm4 clr2 \neq clr3$
END

CONTEXT RINGSETS
  N
CONSTANTS
  n
AXIOMS
  $axm1 n \in N \rightarrowtail\!\!\!\!\twoheadrightarrow N$
  $axm2 \forall s \cdot s \subseteq n[s] \wedge s \neq \varnothing \Rightarrow N \subseteq s$
  $axm3 finite(N)$
THEOREMS $thm1 (\exists x, y \cdot x \in N \wedge y \in N \wedge x \neq y) \Rightarrow id(N) \cap n = \varnothing$
  $thm2 \forall x, s \cdot s \subseteq (n \setminus \{x \mapsto n(x)\})[s] \Rightarrow s = \varnothing$
END

CONTEXT RING1
EXTENDS $ring$
AXIOMS
  $axm1 \exists x, y \cdot x \in N \wedge y \in N \wedge x \neq y$
END

MACHINE $one - shot$
SEES $ring, color$
VARIABLES
  col
INVARIANTS
  $inv1 col \in N \rightarrow color$
EVENTS
  EVENT INITIALISATION
  BEGIN
  $act1 col :\in N \rightarrow color$
END
$/quad$EVENT stableANY
  f WHERE
  $grd1 f \in N \rightarrow color$ $grd2 \forall x \cdot f(x) \notin f[(n \cup n^{-1})[\{x\}]]$WHEN
  $act1 col := f$END
END

MACHINE $rules$
REFINES $one - shot$
SEES $ring1, color$
VARIABLES
col, c
INVARIANTS
$inv1c \in N \rightarrow color$THEOREMS
EVENTS
EVENT INITIALISATION
BEGIN
$act1col, c : |(col' \in N \rightarrow color \wedge c' \in N \rightarrow color \wedge col' = c')$
END
EVENT stableREFINES stableWHEN
$grd1 \forall x \cdot c[\{x\}] \neq c[(n \cup n^{-1})[\{x\}]]$
$grd2 \forall x \cdot c(x) \neq c(n^{-1}(x)) \vee c(x) = c(n(x))$
WITNESSES
$ff = c$
WHEN
$act1col := c$
END
EVENT rule1WHICHIS $convergent$
ANY
x
clr
WHERE
$grd1c[\{x\}] = c[(n \cup n^{-1})[\{x\}]]$
$grd2clr \notin c[\{x\}]$
WHEN
$act1c(x) := clr$
END
EVENT rule2WHICHIS $convergent$
ANY
x
clr
WHERE
$grd1c(x) = c(n^{-1}(x))$
$grd2c(n(x)) \neq c(x)$
$grd3clr \neq c(x)$
$grd4clr \neq c(n(x))$
WHEN
$act1c(x) := clr$
END
VARIANT
$card(\{x | c(x) = c(n^{-1}(x))\})$END

Bibliography

[1] *Boston Scientific: Pacemaker system specification, Technical report, Boston Scientific*, 2007.

[2] J. R. Abrial. Using design patterns in formal devlopments example: A mechanical press controler. journÃ©e scientifique du ppf iaem transversal - dÃ©veloppement incrÃ©mental et prouvÃ© de systÃ¨mes, april 2006.

[3] J.-R. Abrial. *The B book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

[4] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2009. Forthcoming book.

[5] Dominique Cansell and Dominique Méry. *The event-B Modelling Method: Concepts and Case Studies*, pages 33–140. Springer, 2007. See [**?**].

[6] Méry D. Cansell D. and Joris Rehm. *Formal Specification and Development in B*, chapter Time Constraint Patterns for Event B Development, pages 140–154. Lecture Notes in Computer Science. Springer US, 2006. ISSN 0302-9743.

[7] Andreas Furst. Design patterns in event-b and their tool support. Master's thesis, ETH Zurich, 2009.

[8] Aaron Hesselson. *Simplified Interpretations of Pacemaker ECGs*. Blackwell Publishers, 2003. ISBN 1405103728.

[9] David B. Johnson and David A. Maltz. *Mobile Computing*, volume 353 of *The International Series in Engineering and Computer Science*, chapter Dynamic Source Routing in Ad Hoc Wireless Networks, pages 153–181. Springer US, 1996. ISSN 0893-3405.

[10] Jacques Julliand and Olga Kouchnarenko, editors. *B 2007: Formal Specification and Development in B, 7th International Conference of B Users, Besançon, France, January 17-19, 2007, Proceedings*, volume 4355 of *Lecture Notes in Computer Science*. Springer, 2006.

[11] Gary T. Leavens, Jean-Raymond Abrial, Don Batory, Michael Butler, Alessandro Coglio, Kathi Fisler, Eric Hehner, Cliff Jones, Dale Miller, Simon Peyton-Jones, Murali Sitaraman, Douglas R. Smith, and Aaron Stump. Roadmap for enhanced languages and methods to aid verification. In *Fifth Intl. Conf. Generative Programming and Component Engineering (GPCE 2006)*, pages 221–235. ACM, October 2006.

[12] Charles J. Love. *Cardiac Pacemakers and Defibrillators*. Landes Bioscience Publishers, 2006. ISBN 1-57059-691-3.

[13] Jaakko Malmivuo. *Bioelectromagnetism*. Oxford University Press, 1995. ISBN 9780195058239.

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

[15] ProB Tool. The prob animator and model checker for the b method. http://www.stups.uni-duesseldorf.de/ProB/overview.php/.

[16] Jean-Raymond Abrial. *The B book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

[17] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2009.

[18] Projet ANR-RIMEL. Développement d'algorithmes répartis. Livrable RIMEL, LORIA, Janvier/Février 2008.

[19] Projet ANR-RIMEL. Intégration du temps dans le développement incrémental prouvé. Livrable RIMEL, LORIA, Juillet 2008.

[20] Projet ANR-RIMEL. Proof-based design patterns. Livrable RIMEL, LORIA, Juillet 2008.

[21] Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages*. EATCS Textbook in Computer Science. Springer, 2007.

[22] Dominique Cansell and Dominique Méry. *The event-B Modelling Method: Concepts and Case Studies*, pages 33–140. Springer, 2007. See [67].

[23] ClearSy. BART project. http://www.clearsy.com, 2008.

[24] Gary T. Leavens, Jean-Raymond Abrial, Don Batory, Michael Butler, Alessandro Coglio, Kathi Fisler, Eric Hehner, Cliff Jones, Dale Miller, Simon Peyton-Jones, Murali Sitaraman, Douglas R. Smith, and Aaron Stump. Roadmap for enhanced languages and methods to aid verification. In *Fifth Intl. Conf. Generative Programming and Component Engineering (GPCE 2006)*, pages 221–235. ACM, October 2006.

[25] J.-R. Abrial. *The B book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

[26] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*, chapter Modelling the press. Cambridge University Press, 2009.

[27] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2009. Forthcoming book.

[28] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to event-b. *Fundam. Inform.*, 77(1-2):1–28, 2007.

[29] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

[30] Giampaolo Bella. Inductive verification of smart card protocols. *J. Comput. Secur.*, 11(1):87–132, 2003.

[31] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: the three party case. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 57–66, New York, NY, USA, 1995. ACM.

[32] Nazim Benaïssa, Dominique Cansell, and Dominique Méry. Integration of security policy into system modeling. In Julliand and Kouchnarenko [44], pages 232–247.

[33] Nazim Benaissa. Modelling attacker's knowledge for cascade cryptographic protocols. In *ABZ '08: Proceedings of the 1st international conference on Abstract State Machines, B and Z*, pages 251–264, Berlin, Heidelberg, 2008. Springer-Verlag.

[34] Simon Blake-Wilson and Alfred Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 137–158, London, UK, 1998. Springer-Verlag.

[35] Dominique Bolignano. Integrating proof-based and model-checking techniques for the formal verification of cryptographic protocols. In Alan J. Hu and Moshe Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 77–87. Springer, 1998.

[36] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. *Electr. Notes Theor. Comput. Sci.*, 183:39–55, 2007.

[37] Dominique Cansell and Dominique Méry. *The* EVENT B *Modelling Method: Concepts and Case Studies*, pages 33–140. Springer, 2007. see.

[38] Dominique Cansell and Dominique Méry. Incremental parametric development of greedy algorithms. *Electr. Notes Theor. Comput. Sci.*, 185:47–62, 2007.

[39] Dominique Cansell, Dominique Méry, and Joris Rehm. Time constraint patterns for event b development. In Julliand and Kouchnarenko [44], pages 140–154.

[40] Anish. Mathuria Collin. Boyd. *Protocols for Authentication and Key Establishment*. 2003.

[41] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, Mar 1983.

[42] E. Gamma, R. Helm, R. Johnson, R. Vlissides, and P. Gamma. *Design Patterns : Elements of Reusable Object-Oriented Software design Patterns*. Addison-Wesley Professional Computing, 1997.

[43] Rob Jerdonek, Peter Honeyman, Kevin Coffman, Jim Rees, and Kip Wheeler. Implementation of a provably secure, smartcard-based key distribution protocol. In Jean-Jacques Quisquater and Bruce Schneier, editors, *CARDIS*, volume 1820 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 1998.

[44] Jacques Julliand and Olga Kouchnarenko, editors. *B 2007: Formal Specification and Development in B, 7th International Conference of B Users, Besançon, France, January 17-19, 2007, Proceedings*, volume 4355 of *Lecture Notes in Computer Science*. Springer, 2006.

[45] Gary T. Leavens, Jean-Raymond Abrial, Don Batory, Michael Butler, Alessandro Coglio, Kathi Fisler, Eric Hehner, Cliff Jones, Dale Miller, Simon Peyton-Jones, Murali Sitaraman, Douglas R. Smith, and Aaron Stump. Roadmap for enhanced languages and methods to aid verification. In *Fifth Intl. Conf. Generative Programming and Component Engineering (GPCE 2006)*, pages 221–235. ACM, October 2006.

[46] Frank Thomson Leighton and Silvio Micali. Secret-key agreement without public-key cryptography. In *CRYPTO '93: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, pages 456–479, London, UK, 1994. Springer-Verlag.

[47] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In Tiziana Margaria and Bernhard Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[48] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[49] Lawrence C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[50] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[51] Project RODIN. The rodin project: Rigorous open development environment for complex systems. http://rodin-b-sharp.sourceforge.net/.

[52] Paul F. Syverson and Paul C. Van Oorschot. On unifying some cryptographic protocol logics. In *SP '94: Proceedings of the 1994 IEEE Symposium on Security and Privacy*, page 14, Washington, DC, USA, 1994. IEEE Computer Society.

[53] ISO. Information technology - Security techniques - Key management - Part 2: Mechanisms Using Symmetric techniques ISO/IEC 11770-2, 1996. International Standard.

[54] ISO. Information technology - Security techniques - Entity authentication - Part 2: Mechanisms Using Symmetric Encipherment Algorithms ISO/IEC 9798-2, 2nd edition, 1999. International Standard.

[55] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. IEEE Communications magazine, 32(9):33-38, September 1994.

[56] http://media.summitmedicalgroup.com/media/db/ relayhealth-images/nodes.jpg.

[57] A Reseach and Development Needs Report by NITRD. High-Confidence Medical Devices : Cyber-Physical Systems for 21st Century Health Care. http://www.nitrd.gov/About/MedDevice-FINAL1-web.pdf.

[58] J.-R. Abrial. $B^{\#}$: Toward a Synthesis Between Z and B. In D. Bert and M. Walden, editors, *3nd International Conference of B and Z Users - ZB 2003, Turku, Finland*, Lectures Notes in Computer Science. Springer, June 2003.

[59] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2009. Forthcoming book.

[60] J.-R. Abrial and D. Cansell. Click'n prove: Interactive proofs within set theory. In *TPHOL 2003*, pages 1–24, 2003.

[61] R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23(1):49–68, 1979.

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

[63] Dines Bjorner. *Software Engineering 1 Abstraction and Modelling*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. ISBN: 978-3-540-21149-5.

[64] Dines Bjorner. *Software Engineering 2 Specification of Systems and Languages*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. ISBN: 978-3-540-21150-1.

[65] Dines Bjorner. *Software Engineering 3 Domains, Requirements, and Software Design*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. ISBN: 978-3-540-21151-8.

[66] Dines Bjorner. *DOMAIN ENGINEERING Technology Management, Reserach and Engineering*, volume 4 of *COE Research Monograph Series*. JAIST, 2009.

[67] Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages*. EATCS Textbook in Computer Science. Springer, 2007.

[68] Boston Scientific Boston Scientific: Pacemaker system specification, Technical report. 2007.

[69] Dominique Cansell and Dominique Méry. *The event-B Modelling Method: Concepts and Case Studies*, pages 33–140. Springer, 2007. See [67].

[70] Dominique Cansell, Dominique Méry, and Joris Rehm. *Formal Specification and Development in B*, chapter Time Constraint Patterns for Event B Development, pages 140–154. Lecture Notes in Computer Science. Springer US, 2006. ISSN 0302-9743.

[71] ClearSy, Aix-en-Provence (F). *B4FREE*, 2004. http://www.b4free.com.

[72] O. Grumberg E. M. Clarke and D. Peled. *Model Checking*. MIT Press, 1999. ISBN 978-0262032704.

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

[74] E. Gamma, R. Helm, R. Johnson, R. Vlissides, and P. Gamma. *Design Patterns : Elements of Reusable Object-Oriented Software design Patterns*. Addison-Wesley Professional Computing, 1994.

[75] B. S. Goldman, E. J. Noble, J. G. Heller, and D. Covvey. The pacemaker challenge. *CMAJ*, 110(1):28–31, 1974.

[76] Artur Oliveira Gomes and Marcel Vinicius Medeiros Oliveira. Formal specification of a cardiac pacing system. In *FM 2009*, pages 692–707, 2009.

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

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

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

[80] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[81] Gary T. Leavens, Jean-Raymond Abrial, Don Batory, Michael Butler, Alessandro Coglio, Kathi Fisler, Eric Hehner, Cliff Jones, Dale Miller, Simon Peyton-Jones, Murali Sitaraman, Douglas R. Smith, and Aaron Stump. Roadmap for enhanced languages and methods to aid verification. In *Fifth Intl. Conf. Generative Programming and Component Engineering (GPCE 2006)*, pages 221–235. ACM, October 2006.

[82] Insup Lee, George J. Pappas, Rance Cleaveland, John Hatcliff, Bruce H. Krogh, Peter Lee, Harvey Rubin, and Lui Sha. High-confidence medical device software and systems. *Computer*, 39(4):33–38, 2006.

[83] Michael Leuschel and Michael Butler. *ProB: A Model Checker for B*, pages 855–874. LNCS. Springer, 2003.

[84] Charles J. Love. *Cardiac Pacemakers and Defibrillators*. Landes Bioscience Publishers, 2006. ISBN 1-57059-691-3.

[85] Hugo Daniel Macedo, Peter Gorm Larsen, and John Fitzgerald. *Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System Using VDM*, pages 181–197. LNCS. Springer, Los Alamitos, CA, USA, 2008.

[86] Jaakko Malmivuo. *Bioelectromagnetism*. Oxford University Press, 1995. ISBN 0-19-505823-2.

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

[88] Writing Committee Members, Andrew E. Epstein, John P. DiMarco, Kenneth A. Ellenbogen, III Estes, N.A. Mark, Roger A. Freedman, Leonard S. Gettes, A. Marc Gillinov, Gabriel Gregoratos, Stephen C. Hammill, David L. Hayes, Mark A. Hlatky, L. Kristin Newby, Richard L. Page, Mark H. Schoenfeld, Michael J. Silka, Lynne Warner Stevenson, and Michael O. Sweeney. ACC/AHA/HRS 2008 Guidelines for Device-Based Therapy of Cardiac Rhythm Abnormalities: Executive Summary: A Report of the American College of Cardiology/American Heart Association Task Force on Practice Guidelines (Writing Committee to Revise the ACC/AHA/NASPE 2002 Guideline Update for Implantation of Cardiac Pacemakers and Antiarrhythmia Devices): Developed in Collaboration With the American Association for Thoracic Surgery and Society of Thoracic Surgeons. *Circulation*, 117(21):2820–2840, 2008.

[89] ProB. The ProB animator and model checker for the B method. http://www.stups.uni-duesseldorf.de/ProB/overview.php/.

[90] Project RODIN. Rigorous open development environment for complex systems. http://rodin-b-sharp.sourceforge.net/, 2004. 2004–2007.

[91] Joris Rehm. Pattern Based Integration of Time applied to the 2-Slots Simpson Algorithm. In *Integration of Model-based Formal Methods and Tools in IFM'2009*, Düsseldorf Allemagne, 02 2009.

[92] J. Woodcock and R. Banach. The verification grand challenge. 13(5):661–668, 2007.

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