International Journal of Intelligent Computing and Information Science

A METHODOLOGY FOR DESIGNING HIGH CONFIDENCE PATTERN VIA EVENT B

E. K. Elsayed E. E. El-Sharawy A. A. Ibrahim

Mathematical Department., Faculty of s Science, Al-Azhar University, Nasr City, Cairo, Egypt
emankaran10@azhar.edu.eg emankaran10@azhar.edu.eg asmaa_ah@yahoo.com

Abstract: *The correct formal design is an achievement in software engineering, but we faced with challenges to satisfy that. The informal problems types are general or special. The special informal problems depend on the case study and the general informal problems come from an inexperienced designer called anti-patterns". In this paper we discuss these two types on Insulin Infusion Pump (IIP) and sample of UML class diagrams. The proposed approach to formalize IIP is based on using event B. Finally, we could verify that the code generated from the proposed approach is correct and formal to use as a pattern. The accuracy of the proposed verification steps are suitable for using to any systems or medical device. We applied the proposed approach on the sample of eight famous UML class diagrams used as templates. The method ameliorates the proof percentage*

Keywords: *Event-B, Insulin Infusion Pump (IIP), patterns, anti-patterns.*

1. Introduction

Insulin Infusion pumps are medical devices used to drugs delivery to patients at specific rates and in precise amounts. It is necessary to design and manufacture of these medical devices correctly so as for ensuring the safety of patients and defects, but there are many of the informal problems types are general like an anti-pattern or special problems such as in the IIP.

There are many works introduce to detect only special problems like in references [1], [2] and [3]. In addition, there were many works for the general problems detection such as in references [4], [5] and [6], but our proposed method covered all the informal problems (special and general). In addition, Using Formal methods languages like Event-B can be support mathematical reasoning required for detecting this kind of problems.

Formal methods can, and must play an important role to verify the system requirements, and in guaranteeing the correctness, reliability and safety of developing system software. Since software plays an important role in all fields as the medical field, like the FDA, it needs an effective way to evaluate embedded in the devices. That is to certify the systems developed and to guarantee the safe behavior of each system [7, 8, 9]. Many people believe that formal methods have the potential to develop dependable, safe and secure systems that are also more amenable to certification with required features that can be used to certify dependable medical systems [9, 10].

Event- B is a formal method for specifying, modeling and reasoning systems. Event- B is an evolution of the B-Method [11] developed by Jean-Raymond Abrial. A model in Event-b consists of contexts and machines, Contexts contain the static part (types and constants) of a model while Machines contain the dynamic part (variables and events). The modeling elements of a context [12, 13] have four types: sets, constants, axioms, and theorems. A Machine consists of variables, invariants, events, theorems and variants. Variables, v, define the state of a model. All events are Atomic and can be executed only when their guards hold [4]. There are various relationships between contexts and machines. A context can be "extended" by other contexts and "referenced" or "seen" by machines. A Machine can be "refined" by other machines and can reference to contexts as its static part.

A Design pattern is a general repeatable solution to a problem which usually occurring in software design. It's not a finished design which can be converted directly into code. But It is a description or template for how to solve a problem that can be used in various positions. In object-oriented programming, a pattern may include the description of certain objects and object classes to be used, side by side of their attributes and dependencies. The intention of Design Patterns in Event-B is to have a methodological approach to reuse former developments (referred to as patterns) in a new development. The patterns approach is a promising avenue to let inexperienced designers build conceptual models and as a tool for building domain models [14].

In Event-b pattern the proofs of the pattern can be reused too. It was shown that for the special case of a model that does not see any context and its events do not have any parameters, the generation of a refinement of the problem at hand is correct by construction and no proof obligation needs to be generated again. The correctness of the construction relies on a correct matching of the pattern with the problem [15].

Anti-patterns have a negative impact on the comprehension and maintainability of a software system. The designer may introduce anti-patterns in their models because of time pressure, lack of understanding, communication, and–or skills. To develop high-quality models, a designer must have the support of expressive engineering tools such as detection anti-patterns tools. Anti-patterns are defined as design patterns whose purpose is to document common bad practices in software design [16].

The ability to successfully design software projects is highly depending on using patterns without informal problems “Anti-pattern”. This will lead us to believe that having a general tool for verification and validation of a pattern design is important. In software engineering, the levels of detection of anti-patterns are project management level, design level, and code level. Good results have been obtained in all levels. There is a tool under implementation for detecting the project management anti-patterns [17]. Also, there are many papers that propose detection techniques at the design level as proposed in [18], but the code level in [19] focuses on certain types of anti-patterns. The remainder of this paper is organized as follows: Section 2 presents related work. The proposed method phases are presented in Section 3. Our approach is applied in a case study in section 4. Section 5 presents the analysis of results. Finally, we present the Conclusion and future work in section 6.

2. Related Work

In this section, we will present some attempts to improve the pattern quality or solve certain anti-pattern. In 2014 Eman k. [4], proposed an attempt to enhance the software quality by detecting the anti-patterns problems on certain model as ATM model. The proposed approach improved the proof obligation by SMT solver. This method detected only some general anti-patterns in the event B model.

Also, Stoianov, Alecsandar [5] in 2010 introduced a detection method for anti-patterns. That was by a logic-based approach. The main advantage of that was the simplicity of the defining Prolog predicates to describe both structural and behavioral aspects of patterns and anti-patters. That was in code level only. In 2014 [6] authors evaluated different query approaches to locate anti-patterns for refactoring Java programs. In a traditional setup, they use code analysis tool called Columbus to optimize Abstract Semantic Graph and processed by hand-coded visitor queries. On the other hand, an EMF representation was built for the same program model. This approach presents a set of anti-pattern which we also applied it in our case study. But this approach has add locked Technique for detection anti-pattern however, our approach has more general Technique.

Masci, P., Ayoub, A., In 2013 [1] presented the model-based development of user interface behavior of an infusion pump in the Prototype Verification System (PVS). The developed model was verified against relevant safety requirements provided by the FDA. Finally, the PVS code generator was used to produce executable code from the verified specifications.

In 2015 [2] authors presented an incremental proof-based development of IIP. That is by using Event-B modeling language to formalize the given system requirements. Furthermore, the Rodin proof tools are used to verify the correctness of functional behavior, internal consistency checking with respect to safety properties, invariants and events. A generic model for an IIP was developed in the Event-B modeling language to verify the safety requirements related to timing issues as in [3].In addition, there were many works for pattern language. In [20] the author presented a survey of a pattern language for data visualization without specification for any practical applications.

Proposed Method

In this section, we introduce a description of our approach phases for producing a confidence pattern.
The First phase is detecting and correcting all types of informal problems from a model.
That is to formalize a certain model and verify the correctness of it. We choose the IIP devise to check the two types of informal design on it. Then we start by creating an IIP initial model then we finished with a confidence IIP pattern which haven't structure anti-patterns and solved 16 special informal problems by chain of eleven refinements.

The Second phase is increasing the confidence by using event b solver as Satisfiability Modulo Theory (SMT solver). We apply the Automatic theorem prover like SMT solvers with Event-B model to have safeguards for reducing the proving effort and increasing the degree of automation. SMT-solvers are natural candidates to carry out the verification conditions generated by the application of Event-B methods.

The third phase converts model to confidence pattern with saving proof.
Pattern in Event b is to have a systematic approach to reuse the previous developments (referred to as patterns) in a new development. The approach of pattern is a tool for building domain models for the inexperienced designer [15]. The proofs of the pattern can be reused also**.** This phase is important for saving the proofs.

The fourth phase is generation a confidence source code.
The philosophy Code generation means the automatic generation of source code. It can be considered as an “open-loop” refinement step. When there is not static equivalence checking against the previous refinement is possible. It classified into three steps rewriting, Translation and building.

3. Case Study

We implement our case study by using a **RODIN** platform on IIP. Where event-b is supported by a **RODIN** platform [21], an open-source tool implemented in the Eclipse framework, this tool provides an Event-B integrated development environment, automated proof strategies, a model checking and code generation.

Also, Insulin Infusion Pump (IIP) is a small, complex, software-intensive medical device that allows controlling continuous under the skin infusion of insulin to patients. It delivers physiological quantities of insulin between meals and at meal times. An insulin pump composed of the physical pump mechanism, a disposable reservoir and a disposable infusion set. The pump system includes a controller, and a battery. The disposable infusion set includes a cannula for subcutaneous insertion, and a tubing system to interface the insulin reservoir to the cannula. At present, open-loop and closed-loop insulin pumps exist. A closed-loop insulin pump is also known as an artificial pancreas, which automatically monitors and controls the blood glucose level of a patient. In an open-loop insulin pump, patients' needs to monitor the blood glucose level manually. An insulin pump can be programmed to release small doses of insulin continuously (basal), or one shot dose (bolus) before a meal, to control the rise in blood glucose.

The application of the phases is as the following:

The First phase is detecting and correcting all types of informal problems from a model.

1- The special informal problems which depend on the case study as in IIP model. These are as:

Prob1: the basal profile process which the interaction between the user and device doesn’t clear.
Prob2: the pause and resume of the IIP are essential behavior it satisfies the definition of unpredictable events.
prob3: The device can't suspend all active basal delivery or bolus deliver during pump refilling and in the case of system failure.
Prob4: The device doesn't allow undergo a power-on-self-test (POST) whenever the device power is turned on.
Prob5: The device doesn't allow the user to manage system functionalities that related to: stopping the insulin delivery; validating basal profile parameters; reminder management and validating bolus preset parameters.
Prob6: the user doesn't enable to create a food database that can be used to store food or meal descriptions and the carbs associated with them.
Prob7: The device doesn't allow for the user to change parameter setting basal profile, bolus preset, and temporary basal.
Prob8: The device doesn't allow the user to define a basal profile that consists of an ordered set of basal rates of the user.
Prob9: The device can contain several basal profiles, but not only one basal profile can be active at any single point in time.
Prob10: The device doesn't allow the user to override an active basal profile with a temporary basal, without changing the existing basal profile.
Prob11: the active basal profile, can't resume after the temporary basal ends.
Prob12: The user doesn't be able to stop the active normal or extended bolus.
Prob13: The device doesn't maintain a history of basal and bolus dosages over the past n-days.

Prob14: The device doesn't maintain of an electronic log of every operation associated with a user alert
Prob15: The device doesn't provide feedback to the user regarding system and delivery status.
Prob16: The device can't enforce a maximum dosage for the normal bolus or extended bolus.

To solve these special informal problems we present ten incremental refinements of IIP model. These refinements are implemented by Event-B modeling language to formalize the given system. The eleven incremental refinements are briefly described as in different references as [2] [3].
Initial Model (Power Status) includes two phases. Practically, by using the Rodin platform in the preparation phase consisted of the power on/off, pump priming; input settings (user programmed variables).

The insulin infusion phase is the main activation phase of the system. The INFU_START and the INFU_ PROC indicate the main functional behavior of the system. An Event-B context declares enumerated set *power State* defined using axioms (*axm*1 – *axm*2) for power status. An abstract model declares a list of variables defined by invariants (*inv1 – inv3*).

```
CONSTANTS
    on
    off
```

```
AXIOMS
    axm1 : state = {on, off}
    axm2 : on ≠ off
```

We introduce four events to specifying the desired functional behavior for controlling the power status of an IIP. These events include guard(s) for enabling the given action(s), and the actions that define the changes to the states of the power status here, we provide all events related to The preparation phase (POWER ON, INFU START, INFU PROC and POWER OFF).

```
INVARIANTS
inv1 : rate ∈ N
inv2 : power ∈ state
inv3 : power = off ⇒ rate = 0
```

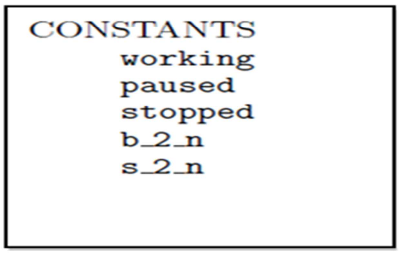
```
Event POWER_ON ≙
    when
        grd1 : power = off
    then
        act1 : power := on
    end
```

```
Event INFU_PROC ≙
    when
        grd1 : power = on
    then
        act1 : rate :∈ N
    end
```

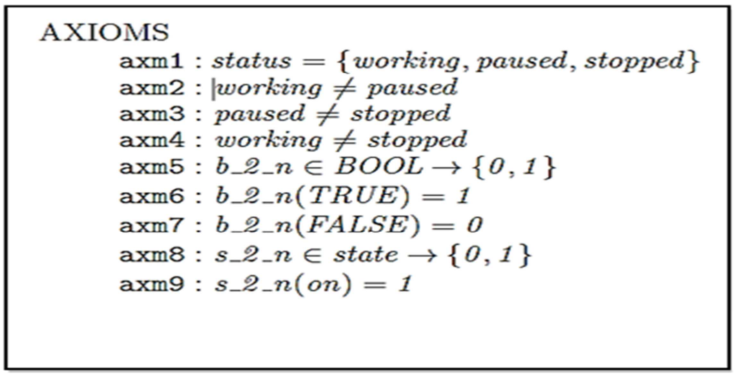
```
Event POWER_OFF ≙
    when
        grd1 : power = on
    then
        act1 : power := off
        act2 : rate := 0
    end
END
```

```
Event  INFU_START

any
   ready

grd1 : ready ∈ BOOL

grd2 : ready = TRUE

grd3 : rate = 0

grd4 : power = on

then
 act1 : rate :∈ N1
end
```

Ref1: This refinement refined the initial model behavior into two phases; one focus on the basal profile setting and the priming process and the other phase refined the infusion process event into several sub events. Practically, in this refinement, we are going to refined The INFU_ PROC event is into several sub events which include pausing, resuming and stopping of the infusion process. Two new events PRIME (priming the pump) and BP_SET (basal profile setting) are introduced to eliminate ready in event INFU_START. INFU_START is refined by adding guards describing that when priming and basal profile setting are done and when the infusion status is stopped. In this refinement, we define an enumerated set and a list of variables to formalize user operations defined by invariants (*inv1 – inv9*).

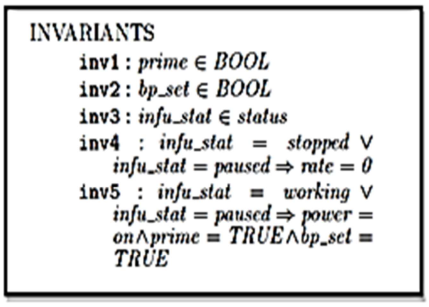


```
CONSTANTS
      working
      paused
      stopped
      b_2_n
      s_2_n
```

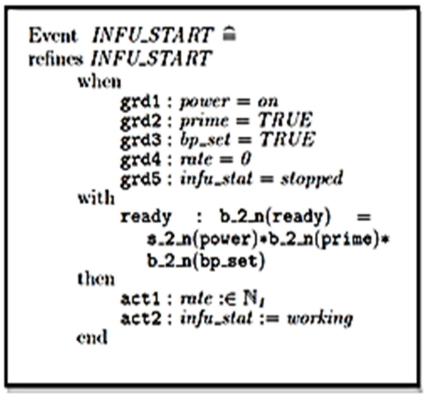


```
AXIOMS
      axm1 : status = {working, paused, stopped}
      axm2 : working ≠ paused
      axm3 : paused ≠ stopped
      axm4 : working ≠ stopped
      axm5 : b_2_n ∈ BOOL → {0, 1}
      axm6 : b_2_n(TRUE) = 1
      axm7 : b_2_n(FALSE) = 0
      axm8 : s_2_n ∈ state → {0, 1}
      axm9 : s_2_n(on) = 1
```

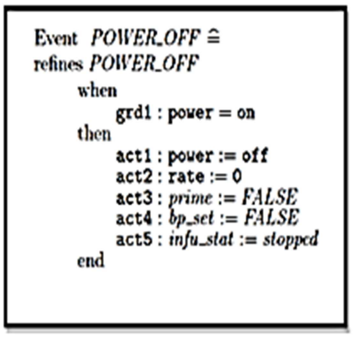
Frist refinement declares a list of variables defined by invariants (*inv1 – inv5*). This refinement step introduces nine events to specify the entire possible INFU_ PROC event. We provide only two events related to refine the INFU_START and POWER_OFF events.



```
INVARIANTS
      inv1 : prime ∈ BOOL
      inv2 : bp_set ∈ BOOL
      inv3 : infu_stat ∈ status
      inv4 : infu_stat = stopped ∨
         infu_stat = paused ⇒ rate = 0
      inv5 : infu_stat = working ∨
         infu_stat = paused ⇒ power =
         on∧prime = TRUE∧bp_set =
         TRUE
```



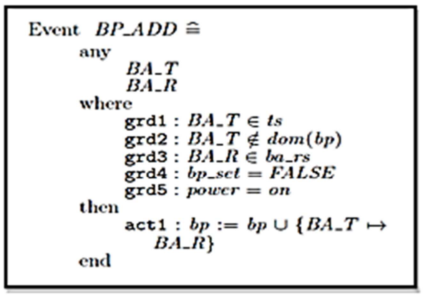
```
Event INFU_START ≙
refines INFU_START
   when
      grd1 : power = on
      grd2 : prime = TRUE
      grd3 : bp_set = TRUE
      grd4 : rate = 0
      grd5 : infu_stat = stopped
   with
      ready : b_2_n(ready) =
         s_2_n(power)∗b_2_n(prime)∗
         b_2_n(bp_set)
   then
      act1 : rate :∈ ℕ₁
      act2 : infu_stat := working
   end
```



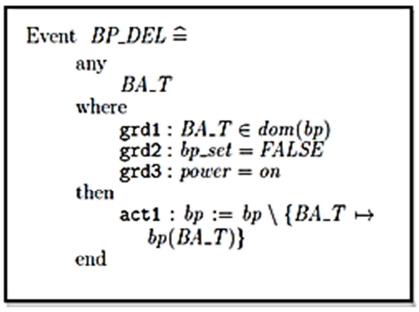
```
Event POWER_OFF ≙
refines POWER_OFF
   when
      grd1 : power = on
   then
      act1 : power := off
      act2 : rate := 0
      act3 : prime := FALSE
      act4 : bp_set := FALSE
      act5 : infu_stat := stopped
   end
```

Ref2: It can be refine the basal profile setting to overcome prob1 by introducing detailed events such as adding, deleting, modifying, and reading basal profile to ameliorate the interaction between patient and device.

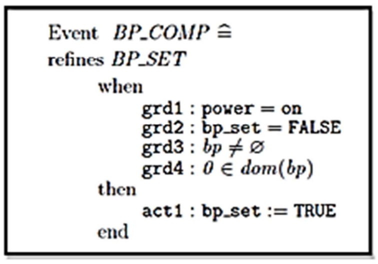
Practically, this refinement introduces a set of operations that is performed between the user and the device for insulin delivering. These operations are adding, deleting, modifying and reading the basal profile. These operations are allowed when an IIP is *on* and we want to deliver an insulin amount in a controlled manner according to the physiological needs of a patient. This refinement introduces five events to specify all the possible user operations related to the given requirements of basal profile. We provide only three events related to The BP_ADD event adds a new pair which does not exist in the basal profile, The BP_DEL event deletes an existing pair from the basal profile and The BP_COMP (basal profile complete) refines the BP_SET by adding guards such as(bp) is not an empty set and 0 is included in the domain of basal profile.



```
Event BP_ADD ≙
    any
        BA_T
        BA_R
    where
        grd1 : BA_T ∈ ts
        grd2 : BA_T ∉ dom(bp)
        grd3 : BA_R ∈ ba_rs
        grd4 : bp_set = FALSE
        grd5 : power = on
    then
        act1 : bp := bp ∪ {BA_T ↦
            BA_R}
    end
```



```
Event BP_DEL ≙
    any
        BA_T
    where
        grd1 : BA_T ∈ dom(bp)
        grd2 : bp_set = FALSE
        grd3 : power = on
    then
        act1 : bp := bp \ {BA_T ↦
            bp(BA_T)}
    end
```



```
Event BP_COMP ≙
refines BP_SET
    when
        grd1 : power = on
        grd2 : bp_set = FALSE
        grd3 : bp ≠ ∅
        grd4 : 0 ∈ dom(bp)
    then
        act1 : bp_set := TRUE
    end
```

Ref3: This refinement introduce the time issues and the corresponding solutions for them prob2 by applying several timing patterns to the infusion process in the IIP. According to the requirements, the main behaviors of the IIP are described in two kinds of insulin infusion processes, called basal and bolus. The insulin infusion process is classified into four groups: basal, temporary basal, bolus and extended bolus. Twenty seven events are introduced to refine the infusion process and use a special event TICK TOCK to control time incrementing in the whole system.

Ref4: This refinement helps to solve both prob3and prob4 by introducing new ten events for specifying the desired functional behavior for controlling the power status of an IIP such as POST_Completed event.

Ref5: This refinement applies to solve each prob5, prob6 and prob8. This refinement displays 35 events to determine all the possible of user operations related to the IIP given problems such as event *CurrActiUserOper_Idle1*that allows a user to create a new basal profile.

Ref6: It introduces to overcoming prob6 and prob7. That is by adding 58 events to model the basal profile management, which contain some functions such as checking the completion of removing task, checking the validity of the selected basal profile, activating basal profile.

Ref7: This refinement can conquer prob10 and prob11.This refinement offer 22 events to model temporary basal profile management. That includes multiple operations like activating, checking validity of entered profile and deactivating the temporary basal profile.

Ref8: This refinement applied to overcome all prob5 and prob12. This refinement submits total 37 events to model the bolus preset Management, which have different subtasks such as creating, activating, checking validity of entered data and removing the bolus preset.

Ref9: This refinement presents to fix prob12 and prob13. It introduces 33 events to model bolus delivery calculation, which includes checking validity of bolus calculation.

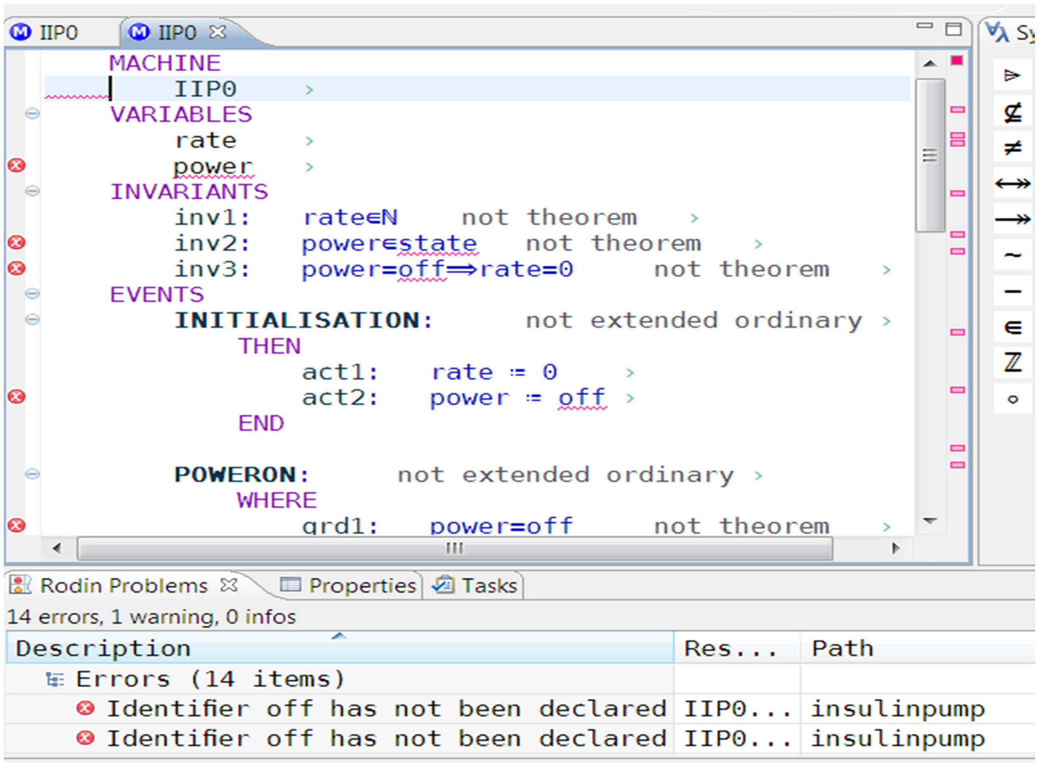
Ref10: This refinement can solve prob14 and prob15 by introducing 28 events to model the reminder management, which includes various operations like creating a new reminder, checking validity of a new entered reminder and removing an old reminder. Finally,

Ref11: This refinement produces a solution for prob16 and prob13. This refinement presents 40 events. Where, 14 events to refine other previously abstractly defined events and 26 events to model the insulin calculator.

2- Detection of the general informal problems called "anti-patterns".

In this part of phase one, we start to detect the anti-patterns from any model. That is by converting it to event-b first. In this case study, we detect the IIP anti-patterns problems by using the Event-b. When you need to construct models, you must be alert to any anti-patterns and solve them. Anti-patterns classify into three main parts; semantic anti-pattern, behavior anti-pattern and structure anti-pattern [4]. Our proposal detects the structure anti-pattern automatically. That is by creating Event-b model and solving the problems.

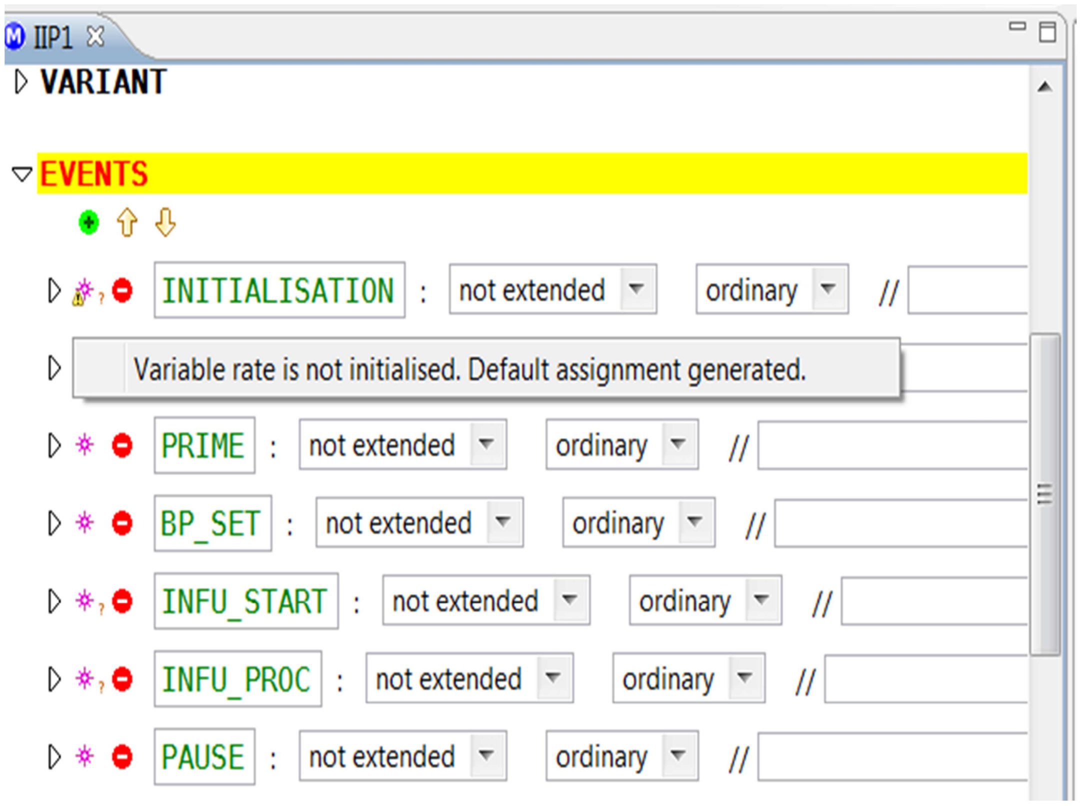
The following examples present some structure anti-patterns. Figure 1 presents the anti-pattern in Event-B machine of IIP model when loose association between Event-B components. The error description list helps designer to know that. Also Figure 2 shows “The default case of the variable not determined anti-pattern” and how Event-B detects it. Another example is “Concatenation to Empty guard anti-pattern” and" unused parameters, anti-pattern "as shown in Figure 3, 4 respectively.



```
MACHINE
        IIP0             >
VARIABLES
        rate             >
        power            >
INVARIANTS
        inv1:   rate∈ℕ      not theorem    >
        inv2:   power∈state    not theorem    >
        inv3:   power=off⇒rate=0       not theorem     >
EVENTS
        INITIALISATION:        not extended ordinary >
            THEN
                act1:    rate ≔ 0     >
                act2:    power ≔ off  >
            END

        POWERON:       not extended ordinary >
            WHERE
                grd1:    power=off      not theorem     >
```

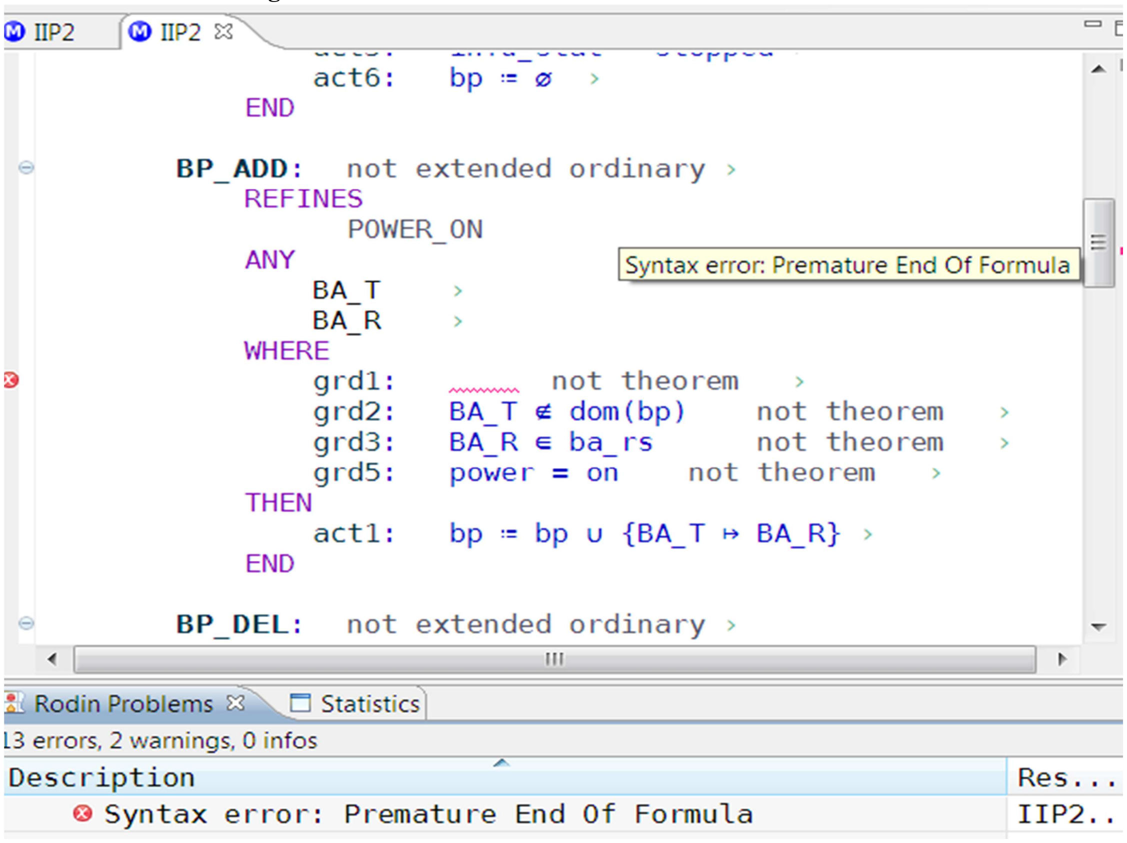
```
14 errors, 1 warning, 0 infos
Description                           Res...   Path
  Errors (14 items)
    Identifier off has not been declared IIP0... insulinpump
    Identifier off has not been declared IIP0... insulinpump
```

Figure1: Missing a relation between Event-B component



```
IIP1
▷ VARIANT
▽ EVENTS
  ▷ INITIALISATION : not extended ordinary //
  ▷ Variable rate is not initialised. Default assignment generated.
  ▷ PRIME : not extended ordinary //
  ▷ BP_SET : not extended ordinary //
  ▷ INFU_START : not extended ordinary //
  ▷ INFU_PROC : not extended ordinary //
  ▷ PAUSE : not extended ordinary //
```

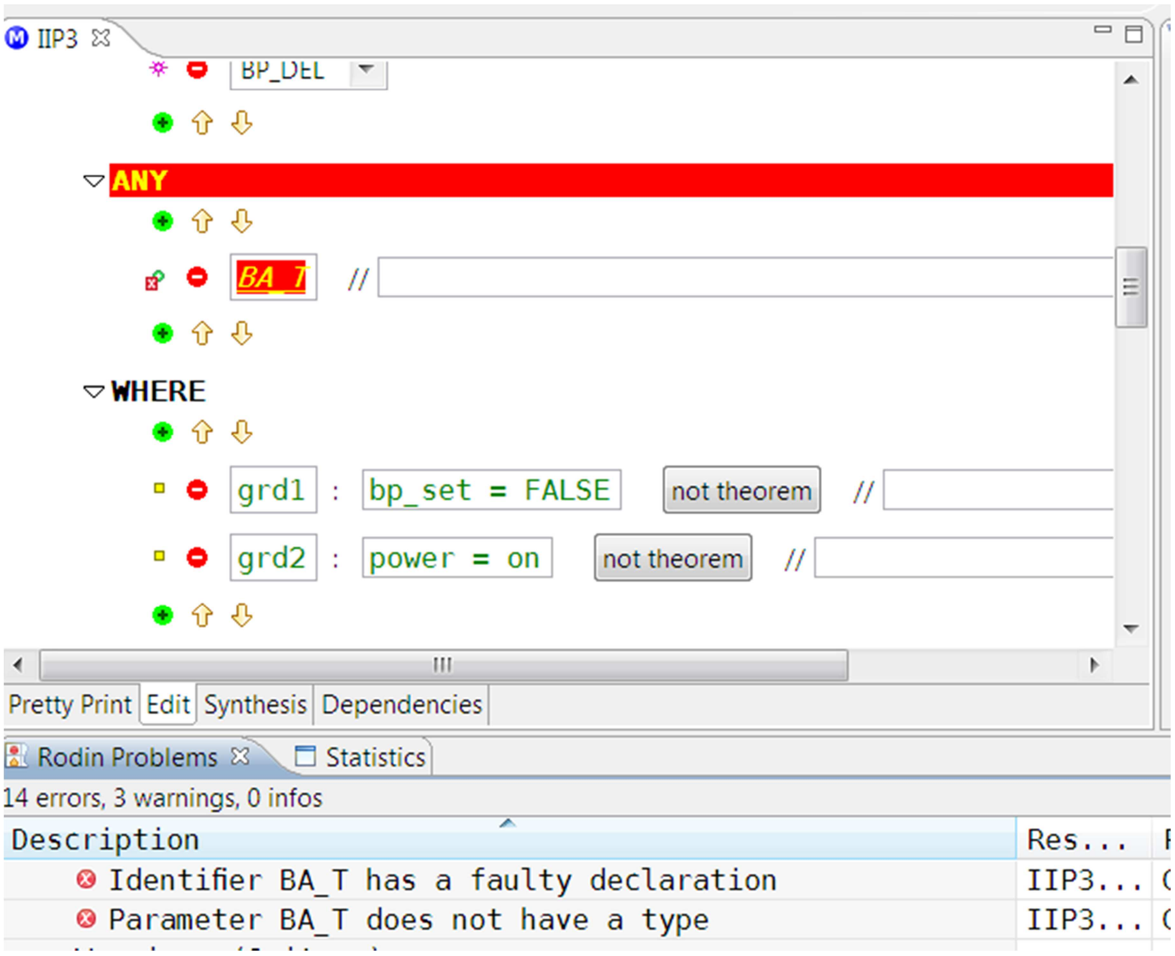
Figure2: The default case of the variable not determined



```
IIP2   IIP2
          act6:    bp ≔ ∅  ›
        END

    BP_ADD:   not extended ordinary ›
        REFINES
               POWER_ON
        ANY                                   Syntax error: Premature End Of Formula
               BA_T      ›
               BA_R      ›
        WHERE
               grd1:          not theorem    ›
               grd2:   BA_T ∉ dom(bp)     not theorem    ›
               grd3:   BA_R ∈ ba_rs       not theorem    ›
               grd5:   power = on     not theorem    ›
        THEN
               act1:   bp ≔ bp ∪ {BA_T ↦ BA_R} ›
        END

    BP_DEL:   not extended ordinary ›
```

```
Rodin Problems   Statistics
13 errors, 2 warnings, 0 infos
Description                                  Res...
  Syntax error: Premature End Of Formula     IIP2..
```

Figure3: Concatenation to Empty guard Anti-pattern


IIP3

BP_DEL

ANY

BA_T //

WHERE

grd1 : bp_set = FALSE not theorem //

grd2 : power = on not theorem //

Pretty Print | Edit | Synthesis | Dependencies

Rodin Problems | Statistics

14 errors, 3 warnings, 0 infos

Description	Res...
Identifier BA_T has a faulty declaration	IIP3...
Parameter BA_T does not have a type	IIP3...

Figure4: Anti-pattern When unused parameters

Also, we detected a general informal problem in another case study called (Hospital UML class diagram). The Hospital UML class diagram contains six classes (Hospital, Booking, Doctor, Patient, Continues and not Continues). The class "Booking" has five attributes and three operations, the class "Doctor" has five attributes and no operations, the class "patient" has no attributes and four operations, the class "Hospital" has two attributes and no operations, the class "Continuous " has two attributes and no operations and finally the class "not Continuous" has two attributes and no operations. The model also has seven associations, some of them with known multiplicity and some not known. That is as shown in Figure5

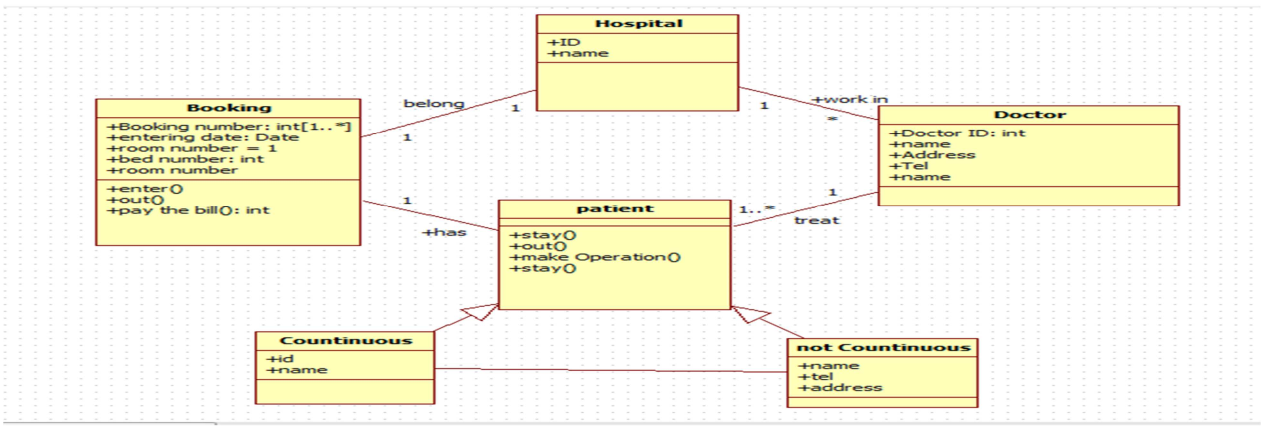
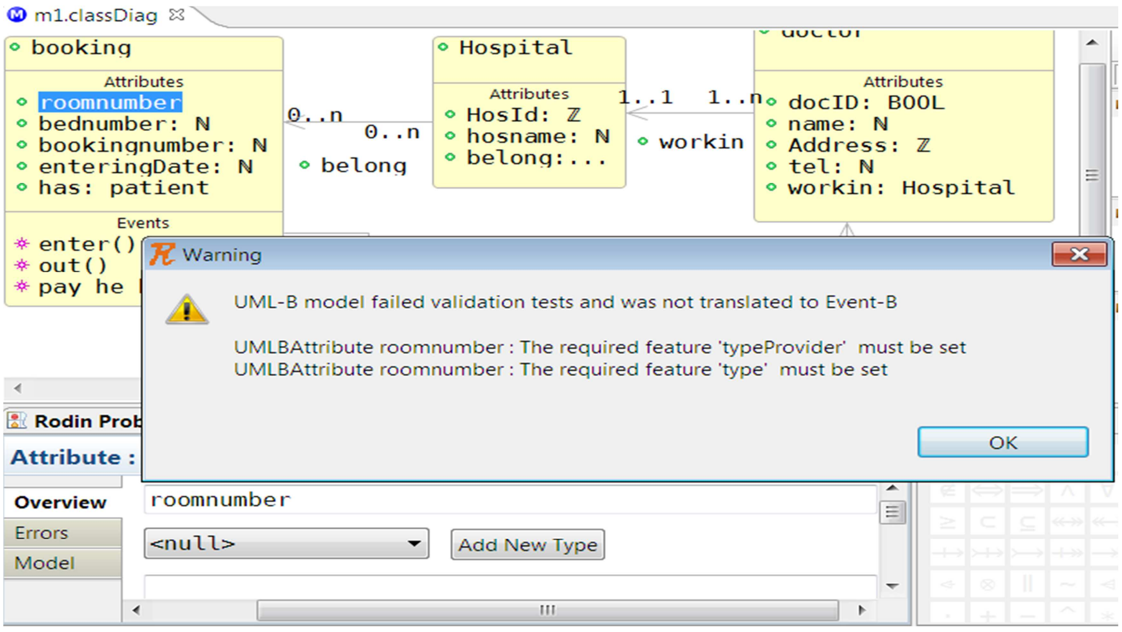


Figure5: Hospital UML class diagram

The UML structure general informal problems (anti-patterns) are detected practically in three levels, some detected during the first level in UML-B or iUML-B when we start by using UML-B plugin certainly. Some detected during the second level "convert UML-B to Event B". Finally, some detected in the third level "proof obligations".

The anti–patterns have been detected in these phases are shown in Figure's 6, 7, 8,9,10 respectively. There are:

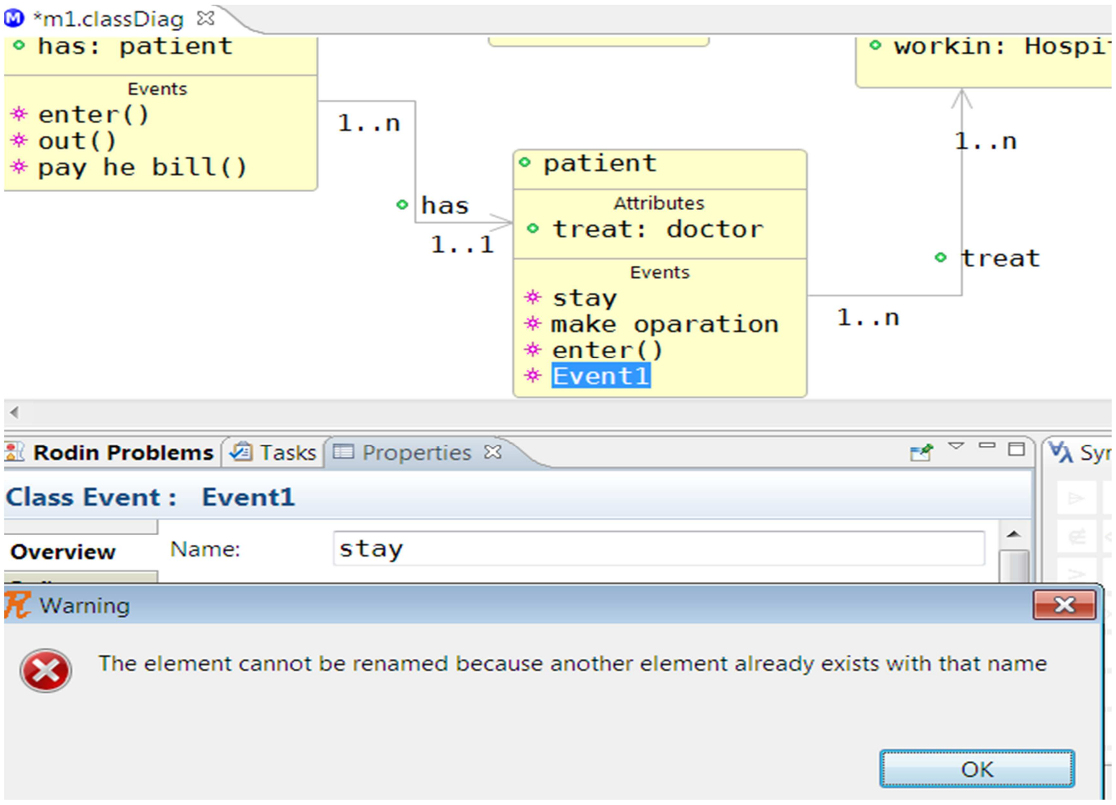
- The attributes of a class haven't data type; attribute (Room number) in class (booking) doesn't have a data type.
- The operations have the same name; class "patient" has two operations with the same name (stay).
- Invalid identifier for the association, the association name must not have any space where association (work in) has space in its name.
- Invalid expression constrain anti-pattern; as shown in the properties slot of the class (Patient) which has an Invalid expression guard of the event "make operation".


Warning

UML-B model failed validation tests and was not translated to Event-B

UMLBAttribute roomnumber : The required feature 'typeProvider' must be set
UMLBAttribute roomnumber : The required feature 'type' must be set

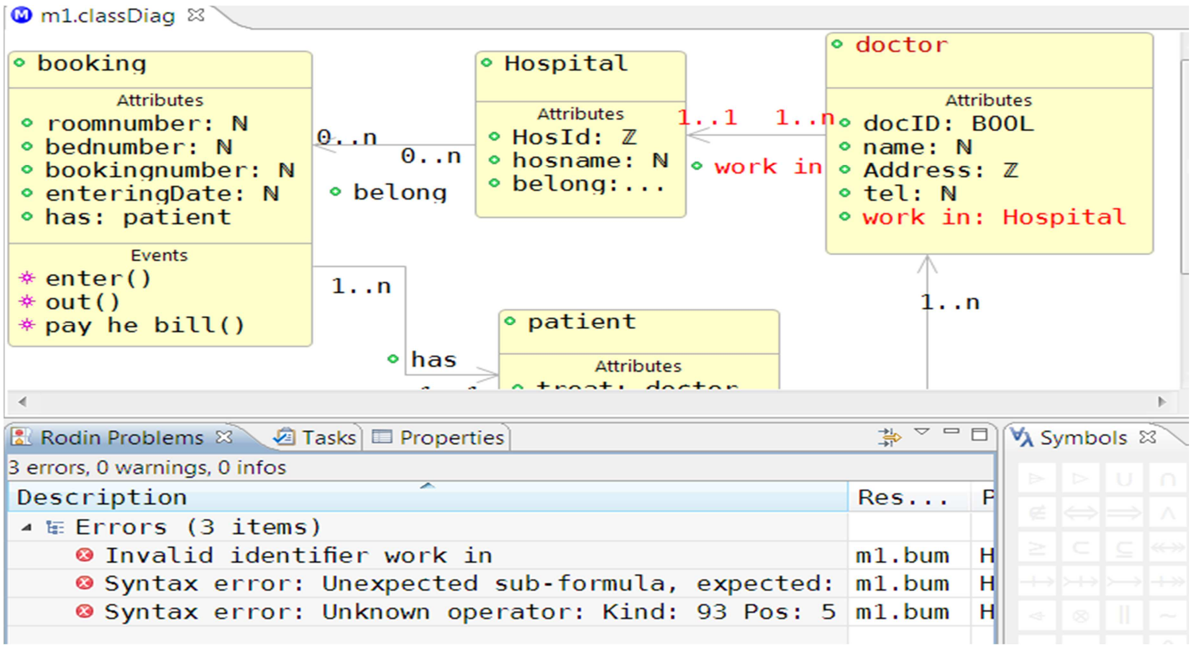
Figure 6: Attribute doesn't have a data type



Warning

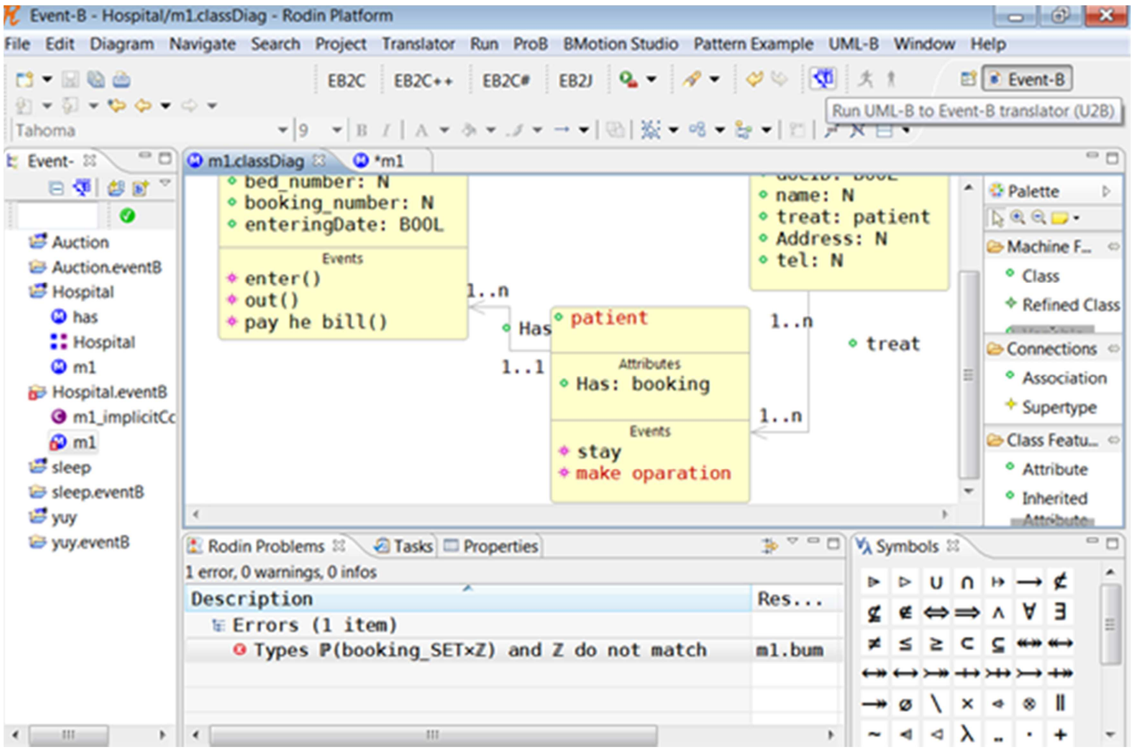
The element cannot be renamed because another element already exists with that name

Figure 7: Two operations have the same name



```
3 errors, 0 warnings, 0 infos
Description                                                     Res...    P
  Errors (3 items)
    Invalid identifier work in                                  m1.bum    H
    Syntax error: Unexpected sub-formula, expected:             m1.bum    H
    Syntax error: Unknown operator: Kind: 93 Pos: 5             m1.bum    H
```

Figure8: Association work in has Invalid identifier

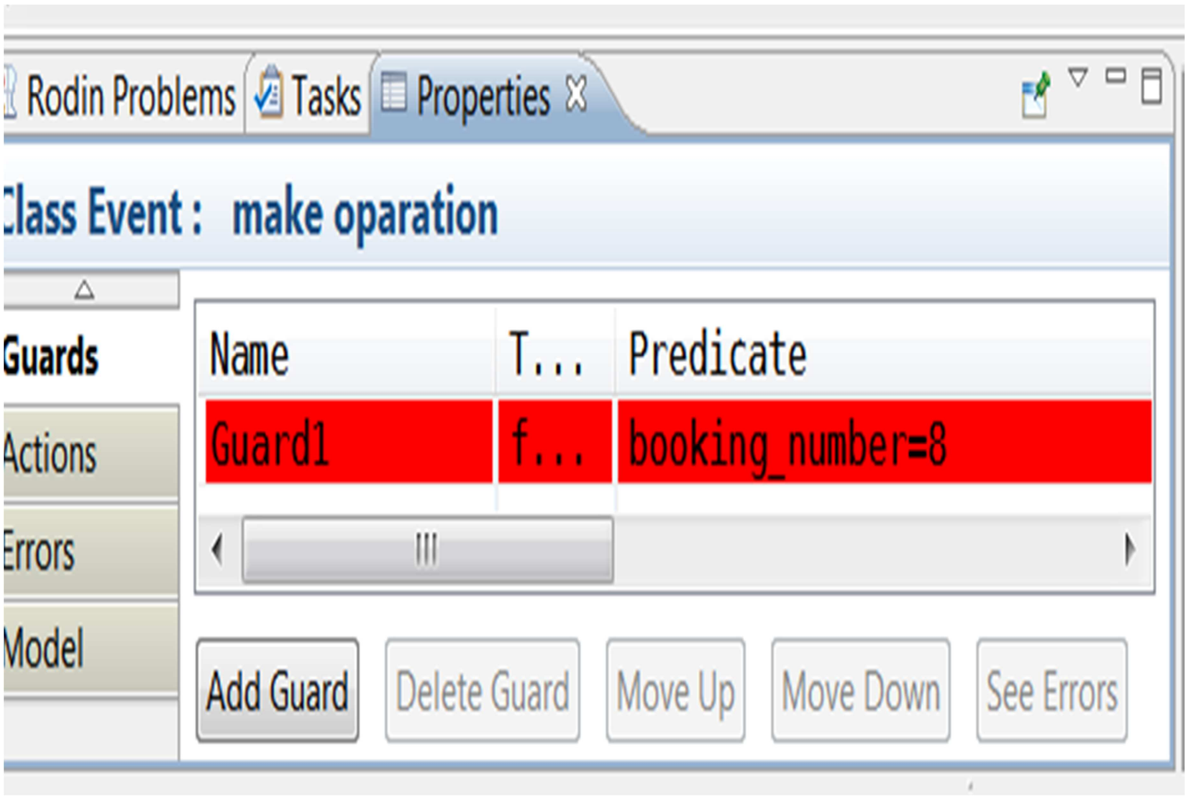


Rodin Problems Tasks Properties

1 error, 0 warnings, 0 infos

Description	Res...
Errors (1 item)	
Types ℙ(booking_SET×ℤ) and ℤ do not match	m1.bum

Figure9: event make operation in class patient

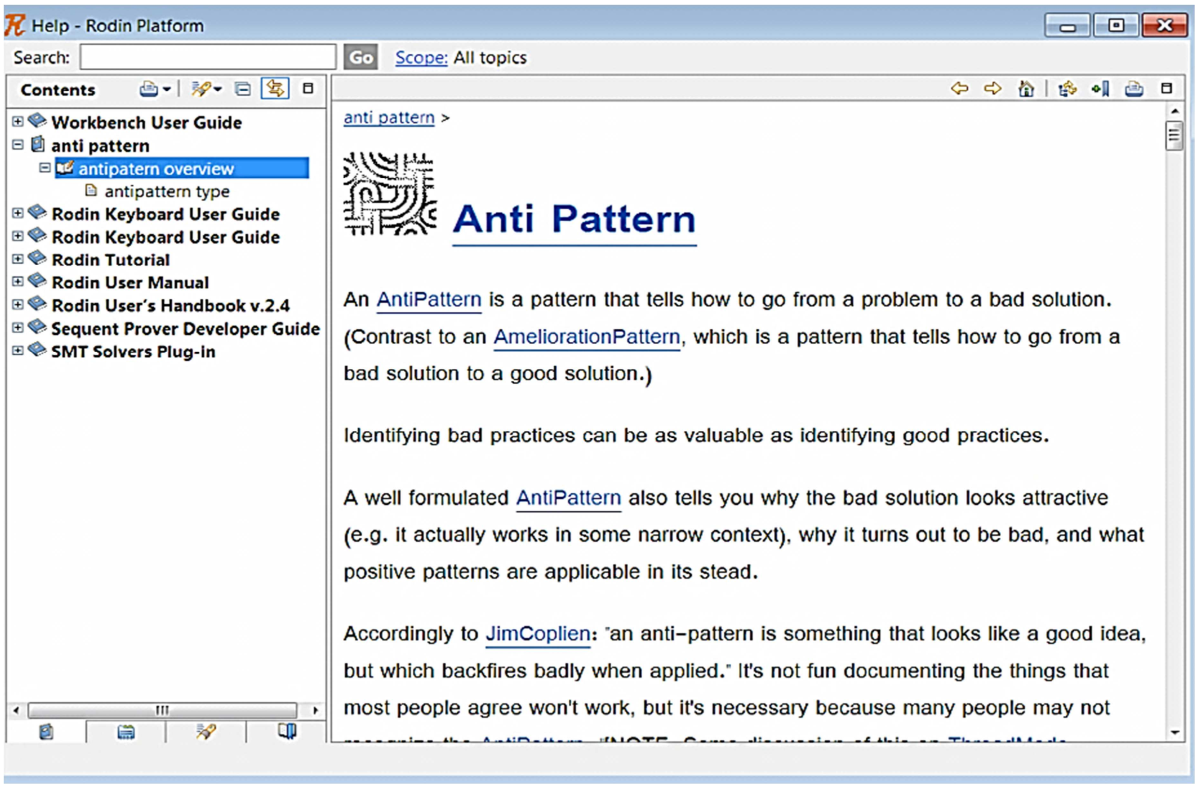


Class Event : make oparation

Name	T...	Predicate
Guard1	f...	booking_number=8

Figure10: Invalid expression guard of the event make operation

From the importance of detecting general anti-patterns we created a new Eclipse plugin in Rodin platform. That is by using the eclipse platform to create this plugin. The plugin has information about all structure anti-patterns to help the users of RODIN. It will only contain documentation files, and then the easiest way to proceed for beginners is to start from the available template. In RODIN Help, the user can see that a new Anti-pattern item has been created in help contents for help menu as shown in Figure11.



An AntiPattern is a pattern that tells how to go from a problem to a bad solution. (Contrast to an AmeliorationPattern, which is a pattern that tells how to go from a bad solution to a good solution.)

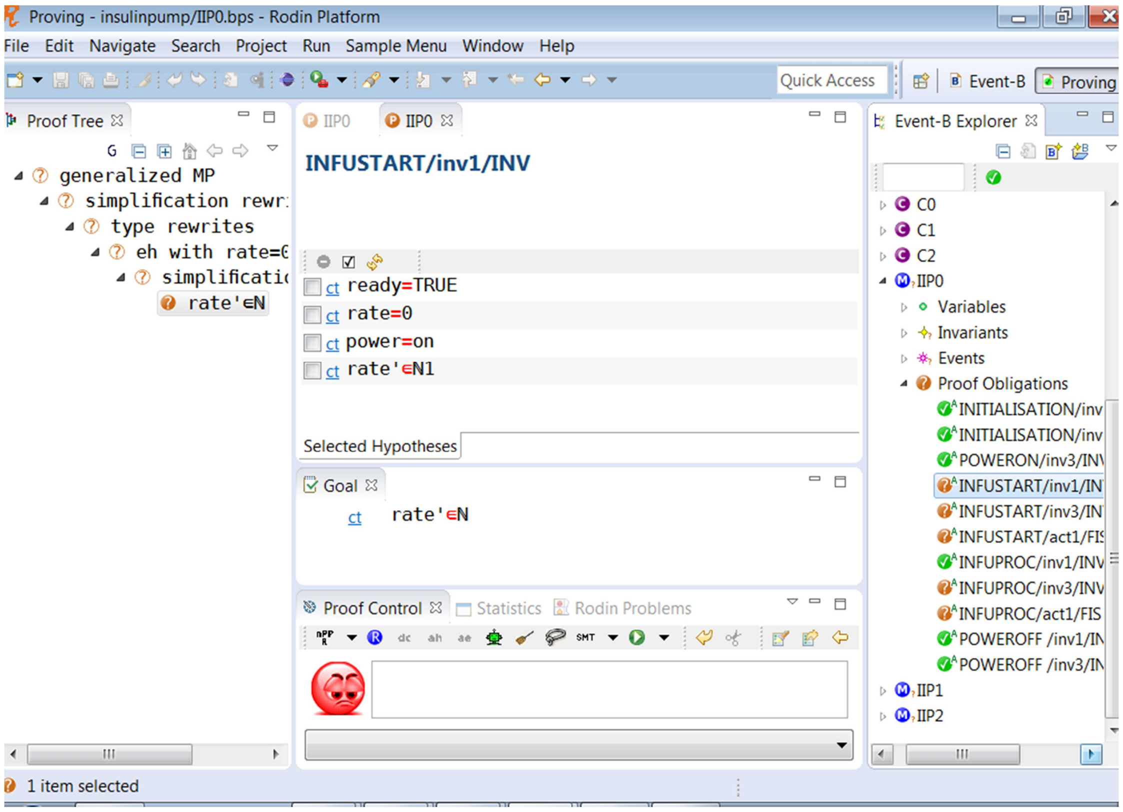
Identifying bad practices can be as valuable as identifying good practices.

A well formulated AntiPattern also tells you why the bad solution looks attractive (e.g. it actually works in some narrow context), why it turns out to be bad, and what positive patterns are applicable in its stead.

Accordingly to JimCoplien: "an anti–pattern is something that looks like a good idea, but which backfires badly when applied." It's not fun documenting the things that most people agree won't work, but it's necessary because many people may not

Figure11: Help contents of RODIN which contain new anti-pattern files.

The second phase is applying SMT solver for increasing the automation degree by decrease the proof obligation. We used SMT-solvers as Automatic verification of proof obligations. The configuration of the plug-in includes a choice of SMT-solvers. It is now available to the formal methods community as an exploratory package through Rodin's official source code repository. Currently, the verification with the SMT-solver has to be activated as shown in Figure 12. The verification was successes as shown in Figure13.



```
generalized MP
  simplification rewr
    type rewrites
      eh with rate=0
        simplificati
          rate'∈N
```

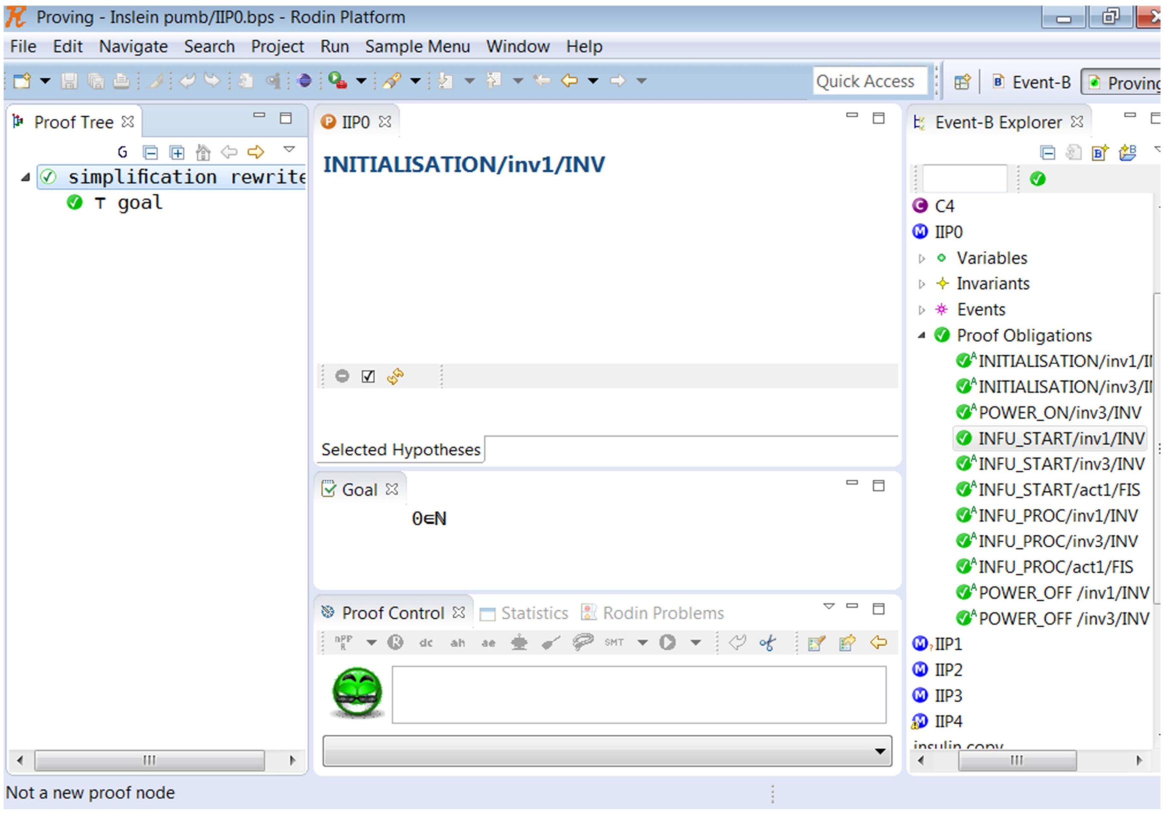
INFUSTART/inv1/INV

```
ct ready=TRUE
ct rate=0
ct power=on
ct rate'∈N1
```

Goal: ct rate'∈N

```
C0
C1
C2
IIP0
  Variables
  Invariants
  Events
  Proof Obligations
    INITIALISATION/inv
    INITIALISATION/inv
    POWERON/inv3/INV
    INFUSTART/inv1/IN
    INFUSTART/inv3/IN
    INFUSTART/act1/FIS
    INFUPROC/inv1/INV
    INFUPROC/inv3/INV
    INFUPROC/act1/FIS
    POWEROFF /inv1/IN
    POWEROFF /inv3/IN
IIP1
IIP2
```

Figure12: the screenshot of the proof before the SMT proof, the button is active and the status is" not proved



```
simplification rewrite
  ⊤ goal
```

INITIALISATION/inv1/INV

Goal: 0∈N

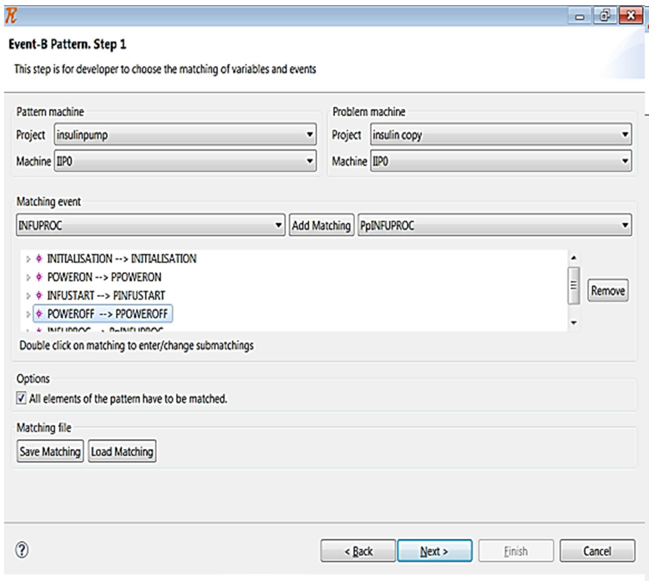
```
C4
IIP0
  Variables
  Invariants
  Events
  Proof Obligations
    INITIALISATION/inv1/I
    INITIALISATION/inv3/I
    POWER_ON/inv3/INV
    INFU_START/inv1/INV
    INFU_START/inv3/INV
    INFU_START/act1/FIS
    INFU_PROC/inv1/INV
    INFU_PROC/inv3/INV
    INFU_PROC/act1/FIS
    POWER_OFF /inv1/INV
    POWER_OFF /inv3/INV
IIP1
IIP2
IIP3
IIP4
insulin copy
```

Figure13: The screenshot of the proof after a successful proof, the button has been dis-activated and the status is "proved"

The third phase converts model to confidence pattern with saving proof.

Practically, in this phase we use "pattern plugin" in RODIN platform. That is for more abstract solution. The screenshot in figure(s) 14, 15,16 and 17 present the convert steps. Where, figure8 is illustrates the matching step between the problem and the specification. This phase contains a dialog for the developers to choose the matching between variables and events.

Figure 9 presents the syntax checking of the matching provided by the user in the previous step. Figure 10 presents the incorporating of the refinement of the pattern into the development. And Figure 11 presents the renaming pattern of both variables and events.



Event-B Pattern. Step 1

This step is for developer to choose the matching of variables and events

Pattern machine

Project: insulinpump

Machine: IIP0

Problem machine

Project: insulin copy

Machine: IIP0

Matching event

INFUPROC | Add Matching | PpINFUPROC

- INITIALISATION --> INITIALISATION
- POWERON --> PPOWERON
- INFUSTART --> PINFUSTART
- POWEROFF --> PPOWEROFF

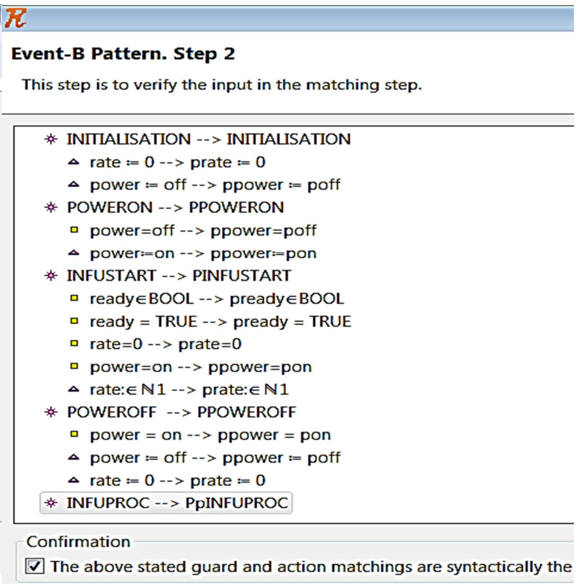
Double click on matching to enter/change submatchings

Options

☑ All elements of the pattern have to be matched.

Matching file

Figure14 First step. Matching



Event-B Pattern. Step 2

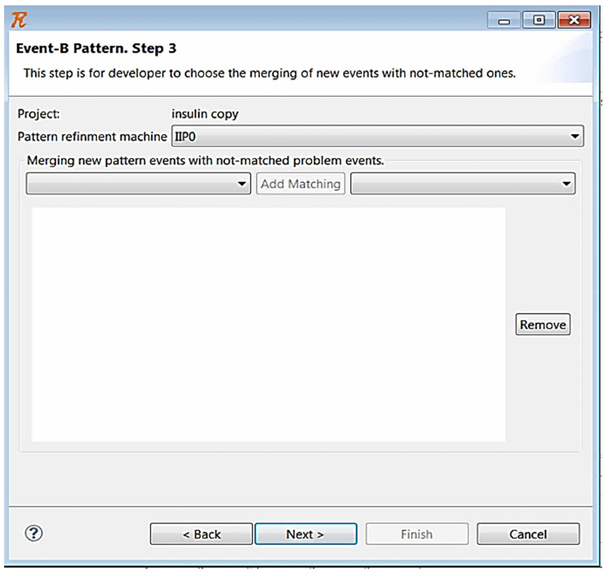
This step is to verify the input in the matching step.

- INITIALISATION --> INITIALISATION
  - rate ≔ 0 --> prate ≔ 0
  - power ≔ off --> ppower ≔ poff
- POWERON --> PPOWERON
  - power=off --> ppower=poff
  - power≔on --> ppower≔pon
- INFUSTART --> PINFUSTART
  - ready∈BOOL --> pready∈BOOL
  - ready = TRUE --> pready = TRUE
  - rate=0 --> prate=0
  - power=on --> ppower=pon
  - rate:∈ ℕ1 --> prate:∈ ℕ1
- POWEROFF --> PPOWEROFF
  - power = on --> ppower = pon
  - power ≔ off --> ppower ≔ poff
  - rate ≔ 0 --> prate ≔ 0
- INFUPROC --> PpINFUPROC

Confirmation

☑ The above stated guard and action matchings are syntactically the

Figure15 Second step. Syntax Checking



Event-B Pattern. Step 3

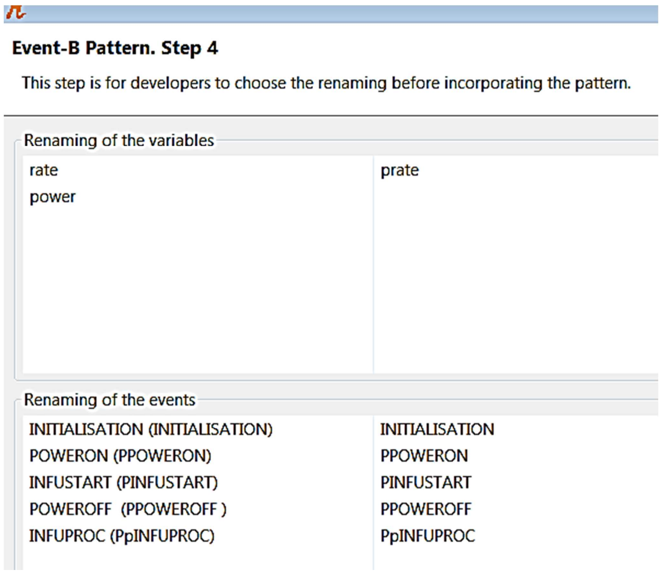
This step is for developer to choose the merging of new events with not-matched ones.

Project: insulin copy

Pattern refinement machine: IIP0

Merging new pattern events with not-matched problem events.

Figure 16 Third step. Incorporating



Event-B Pattern. Step 4

This step is for developers to choose the renaming before incorporating the pattern.

Renaming of the variables

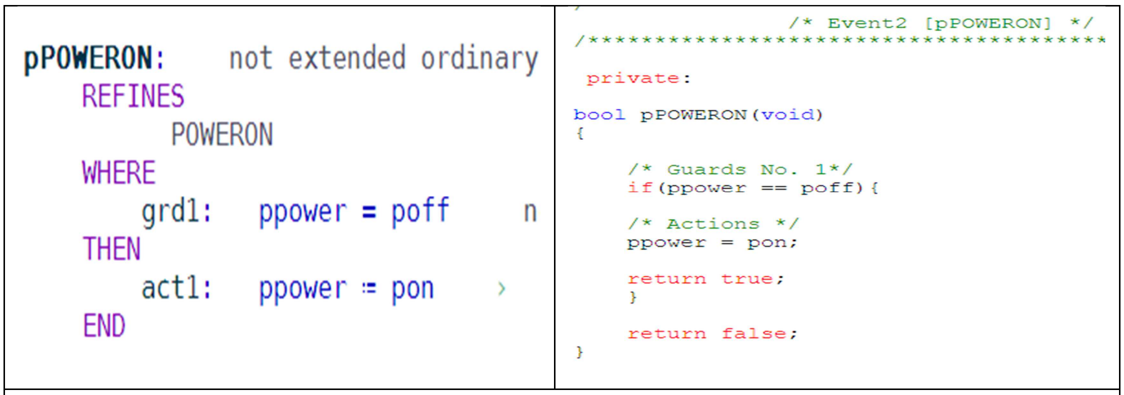
rate	prate
power	

Renaming of the events

INITIALISATION (INITIALISATION)	INITIALISATION
POWERON (PPOWERON)	PPOWERON
INFUSTART (PINFUSTART)	PINFUSTART
POWEROFF (PPOWEROFF)	PPOWEROFF
INFUPROC (PpINFUPROC)	PpINFUPROC

Figure 17 Fourth step. Renaming

The fourth phase is Automatic C++ Code Generation for IIP Pattern Model. Once the pattern has been converted manually, all selected events- without explicit- was translated automatically using the EB2C++ plugin for the RODIN Platform. A single C++ file is produced for each machine. C++ code generation has a different phase of the translation process. An Event-b pattern is initially restated in an easily translatable subset of the notation “Rewrite Phase”. C++ code is then automatically generated from the pattern through a suitable tool “Translation Phase”. Finally “Build Phase” adds support functions and the compiler of the source code language. After applying three phases for generating C++ code from IIP algorithm in Event-B model we obtained the code as in Figure 18.



```
pPOWERON:    not extended ordinary
    REFINES
        POWERON
    WHERE
        grd1:   ppower = poff
    THEN
        act1:   ppower ≔ pon
    END
```

```
                          /* Event2 [pPOWERON] */
/**************************************************
 private:
bool pPOWERON(void)
{
    /* Guards No. 1*/
    if(ppower == poff){

    /* Actions */
    ppower = pon;

    return true;
    }

    return false;
}
```

Figure18: Example of Event PPOWERON translated in C++ code

4. Analysis of result

In this section, we analysis the results of our proposed method as in the following:-

In RODIN platform, a model can’t prove full automatic, so we unable to generate code from the Rodin event B model directly. But by using the SMT solver help for increasing the proof obligation automatically and save it. By using SMT-Solver approach the proof raised to (62%) proof instead of (44%) that obtained by applying Rodin only as shown in table1 that present the Proof statistics of Insulin Infusion Pump (IIP) model after solving all informal and anti-patterns from the model and using SMT solver.

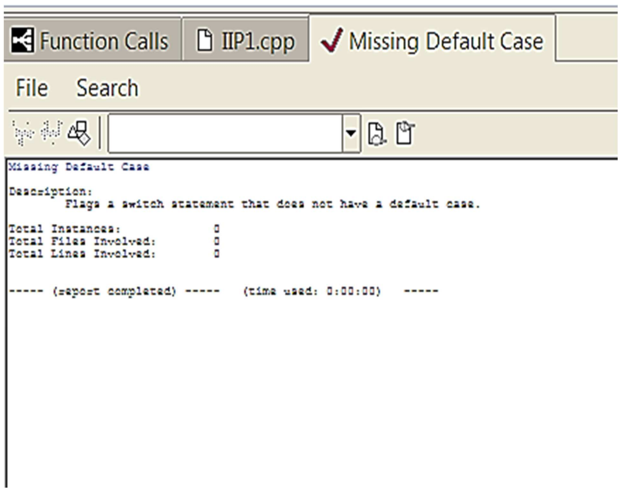
POs **Methods / POs**	**Total** POs	**Auto**	**Interactive (SMT)**	**Proved** POs **%**	**Undercharged**
Rodin	214	96	0	96(44%)	118
SMT	214	96	38	134(62%)	80

Table1. Proof Obligation analysis of IIP pattern.

Now, to verify and ensure that there are not anti-patterns problems were found in our IIP pattern, we used reverse engineering tools like Imagix4D [22] which check all properties of C++ generated code (according to the fourth phase**).** Then we produce a list of reports such as:

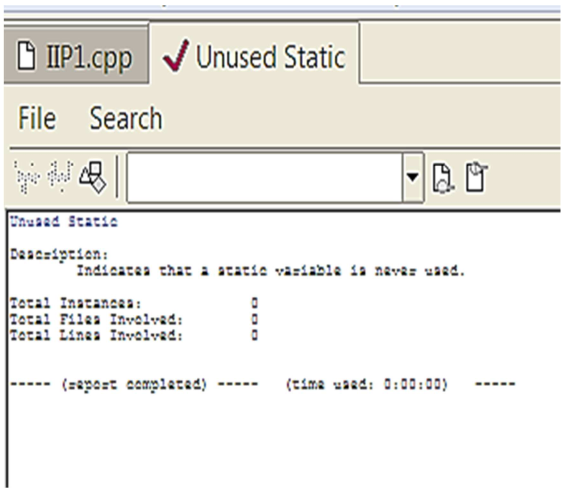
- The report "Missing Default Case" in our code like switch statement that does not have a default case in figure 19.

- The report on the "Variables That Are Never Used" like unused global, static and local variables, which, depending on the option selected such as variables that are read but never set by any functions figure 20.
- The report on "Missing Return Type" such as function declaration that does not specify the return type
- And the report about "Un-terminated Case" likes a case in a switch statement that is not terminated by an explicit transfer of control, such as a break statement.



```
Missing Default Case

Description:
        Flags a switch statement that does not have a default case.

Total Instances:            0
Total Files Involved:       0
Total Lines Involved:       0


----- (report completed) -----   (time used: 0:00:00)   -----
```

Figure19: Imagix4D Report for uninitialized variables



```
Unused Static

Description:
        Indicates that a static variable is never used.

Total Instances:            0
Total Files Involved:       0
Total Lines Involved:       0


----- (report completed) -----   (time used: 0:00:00)   -----
```

Figure20:Imagix4D Report for unused variables

We also detect and correct the anti-patterns on eight UML class diagrams. These patterns are ATM UML class diagram [23], Library and Android UML class diagrams [24], Hasp UML class diagram [25], Seminar, Order and Auction UML class diagrams [26] and Furniture UML class diagram [27]). That is needed to start the method by converting phase to translate UML class diagrams to event-b models. "iUML-B" class diagrams plugin in RODIN Platform [28] is suitable for that. The table 2 presents the 229 general anti-patterns which classified into three groups on the sample of eight class diagrams. The three groups are structure anti-patterns of attributes, structure anti-patterns of class and structure anti-patterns of association. The highest detected number of structure anti-patterns is in association group.

	Anti-patterns	ATM	Order	Seminar	Auction	Library	HASP Java	Furniture system	Android system	
1	structure anti-patterns of attributes	31	2	5	5	26	3	4	-	76
2	structure anti-patterns of class	3	3	-	1	1	-	-	-	8
3	structure anti-patterns of association	14	16	24	9	29	8	14	31	145
Total		48	21	29	15	56	11	18	31	229

Table.2: number of Occurrences of structure Anti-patterns in the UML patterns

5. Conclusions and Further Works

The proposed method is used to verify a medical model such as insulin pump. That is by using formal method to ensure the correctness and stability during proof stage. Using pattern after removing the anti-patterns give us two properties reusability and save the effort of modeling. The code was generated after detection anti-patterns are more confident.

In the future, we are going to correct the anti-patterns automatically. Also we will create a plugin in RODIN to detect and fix the anti-patterns automatically. Also, we will make a comparative study between our method and the methods based on ontological analysis.

References

1. Masci, P., Ayoub, A., Curzon, P., Lee, I., Sokolsky, O., Thimbleby, H.: Model-based development of the generic PCA infusion pump user interface prototype in PVS. In: Bitsch, F., Guiochet, J., Kaˆaniche, M. (eds.) SAFECOMP. LNCS, vol. 8153, pp. 228–240. Springer, Heidelberg (2013).
2. Singh N.K., Wang H., Lawford M., Maibaum T.S.E., Wassyng A. Stepwise Formal Modelling and Reasoning of Insulin Infusion Pump Requirements. In: Duffy V. (eds) Digital Human Modeling. Applications in Health, Safety, Ergonomics and Risk Management: Ergonomics and Health. DHM 2015. Lecture Notes in Computer Science, vol 9185. Springer, Cham (2015).
3. Xu, H., Maibaum, T.: An Event-B approach to timing issues applied to the generic insulin infusion pump. In: Liu, Z., Wassyng, A. (eds.) FHIES 2011. LNCS, vol. 7151, pp. 160–176. Springer, Heidelberg (2012)
4. Eman K. Elsayed, converting UML class diagram with anti-pattern problems into verified code relying on event-b, AIML journal, ISSN 1687-4846, Volume 14, issue 1, ICGST LLC, USA, August, (2014).
5. Stoianov, Alecsandar. "Detecting patterns and anti-patterns in software using Prolog rules." Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on. IEEE, (2010).
6. Zoltán Ujhelyi, Ákos Horváth, Dániel Varró, Anti-pattern Detection with Model Queries: A Comparison of Approaches, vol. 00, no. , pp. 293-302, IEEE, (2014).
7. Chen, Y., Lawford, M., Wang, H., Wassyng, A.: Insulin pump software certification. In: Gibbons, J., MacCaull, W. (Eds.) FHIES 2013. LNCS, vol. 8315, pp. 87–106. Springer, Heidelberg (20114)
8. Keatley, K.L.: A review of the FDA draft guidance document for software validation: guidance for industry. Qual. Assur. 7(1), 49–55 (1999)
9. Lee, I., Pappas, G.J., Cleaveland, R., Hatcliff, J., Krogh, B.H., Lee, P., Rubin, H., Sha, L.: High-confidence medical device software and systems. Computer 39(4), 33–38 (2006)
10. M´ery, D., Singh, N.K.: Real-time animation for formal specification. In: Aiguier, M., Bretaudeau, F., Krob, D. (Eds.) Complex Systems Design and Management, pp. 49–60. Springer, Berlin Heidelberg (2010).
11. Abrial, Jean-Raymond, and Jean-Raymond Abrial. The B-book: assigning programs to meanings. Cambridge University Press, 2005.
12. BOITEN, EERKE, and Jean-Raymond Abrial. "Modeling in Event-B-System and Software Engineering." Journal of Functional Programming 22.2 (2012): 217.

13. Abrial, Jean-Raymond, et al. "An open extensible tool environment for Event-B." Formal Methods and Software Engineering. Springer Berlin Heidelberg. 588-605, (2006) .
14. Xiaohong Yuan, X. & Fernandez, E.," Patterns for business-to-consumer ecommerce applications", International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3, pp 1 (2011).
15. Eman Elsayed, Gaber El-Sharawy and Enas El-Sharawy," Integration of automatic theorem provers in event-b patterns",International Journal of Software Engineering & Applications (IJSEA), Vol.4, No.1, January (2013).
16. W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis, 1st ed. John Wiley and Sons, March 1998.
17. Stamelos, I. (2010),Software project management anti-patterns, Journal of Systems and Software 83 (1), 52-59.
18. Fourati, R., Bouassida, N., & Abdallah, H. B. (2011). A metric-based approach for anti-pattern detection in uml designs". In Computer and Information Science 2011 (pp. 17-33). Springer Berlin Heidelberg.
19. Maiga, A., Ali, N., Bhattacharya, N., Sabane, A., Gueheneuc, Y. G., & Aimeur, E. (2012, October). SMURF: a SVM-based incremental anti-pattern detection approach. In 2012 19th Working Conference on Reverse Engineering (pp. 466-475). IEEE.
20. Niklas Elmqvist, Ji Soo Yi, "Patterns for visualization evaluation", Information Visualization , Vol. 14 (3) 250–269 , (2015).
21. Project RODIN: rigorous open development environment for complex systems (2004). http:// rodin-b sharp.sourceforge.net/
22. imagix4D reverse engineer tool available at https://www.imagix.com/products/source-code-analysis.html.
23. ATM class diagram available at https://www.lucidchart.com/pages/class-diagram-for-ATM-system-UML, access at Jan. 2017.
24. Library and Android class diagrams available at http://www.uml-diagrams.org/class-diagrams-overview.html, access at Jan. 2017.
25. Hasp class diagram available at http://www.uml-diagrams.org/software-licensing-class-diagram-example.html?context=cls-examples, , access at Jan. 2017.
26. Seminar, Order and Auction class diagrams available at http://creately.com/diagram/example/gsxncbybt/Seminar+Class+Diagram , access at Jan. 2017.
27. Furniture class diagram available at https://www.gliffy.com/go/html5/launch?app=1b5094b0-6042-11e2-bcfd-0800200c9a66&templateId=4218693, access at Jan. 2017.
28. Event B class diagrams iUML-B ver. 1.2.0 released at Des. 2015 https://sourceforge.net/projects/Rodin-b-harp/files/