Stepwise formal modeling and verification of Self-Adaptive systems with Event-B. The automatic rover protection case study

Neeraj Kumar Singh, Yamine Aït-Ameur, Marc Pantel, Arnaud Dieumegard, Eric Jenn

Stepwise Formal Modeling and Verification of Self-Adaptive systems with Event-B. The Automatic Rover Protection case study

Neeraj Kumar Singh,Yamine Aït-Ameur and Marc Pantel
INPT-ENSEEIHT / IRIT, University of Toulouse, France
Email: {nsingh, yamine, marc.pantel}@enseeiht.fr

Arnaud Dieumegard and Eric Jenn
Institute of Technology Antoine de Saint Exupéry, France
Email: {arnaud.dieumegard, eric.jenn}@irt-saintexupery.com

***Abstract*—For a long time, formal methods have been effectively applied to design and develop safety-critical systems to ensure safety and the correctness of desired functional behaviors through formal reasoning. The development of high confidence self-adaptive autonomous systems, such as Automatic Rover Protection(ARP), is one of the challenging problems in the area of verified software that needs formal reasoning and proof-based development. In this paper, we propose a methodology that reveals the issues involved in the formal modeling and verification of self-adaptive autonomous systems using *correct by construction approach*. This work also provides a set of guidelines for tacking the different issues to avoid collision by preserving the local and global properties of an autonomous system. We cater for the specification of functional requirements, timing requirements, spatial and temporal behavior, and safety properties. We present a refinement strategy, modeling patterns to capture the essence of a self-adaptive autonomous system, and a substantial example based approach on an industrial case study: TwIRTee. For developing the formal models of autonomous system, we use the Event-B modeling language and associated Rodin tools to check and verify the correctness of required system behavior and internal consistency under the given safety properties.**

***Index Terms*—Automatic Rover Protection (ARP), Event-B, refinement, formal methods, verification, validation.**

I. Introduction

Embedded systems are pervasive in the electronic systems, such as medical device, avionics, transportation, consumer electronics, communication systems, that assist to our daily lives. Therefore, there is an increasing interest for developing safe and dependable embedded systems to provide system reliability, safety, performance and autonomy. An autonomous self-adaptive software system runs in intricate environment with frequent and unexpected changes, where such system always modifies its functional behavior in order to adapt new changes within the system [1]. Such systems are highly complex and tightly integrated with executional environment. There are several approaches to develop such systems, but none of them is sufficient. There is a crucial need for new methods to support the development of self adaptive critical systems to guarantee the functional correctness of the developed systems. The availability of new methods and tools could significantly save time and cost for developing such systems.

Traditional techniques, such as testing and simulation become old fashioned approach that are time consuming and expensive to deploy in the development of embedded systems and to meet the public demands with required qualities. There is a crucial need to use better techniques to meet the desired qualities by saving cost and time for developing and certifying the embedded systems. From many years, formal methods have been applied successfully to design and develop the embedded systems. In particular, they have been used to check the system requirements related to the functional behavior of a given system. For instance, it has been used in the development of the critical systems like medical, avionics, distributed system and automotive [2], [3], [4], [5], [6].

Refinement is a key ingredient for handling the development of large complex systems. It supports the modeling of the whole system incrementally. Each incremental step adds details to existing functionalities while preserving the already established safety properties. Moreover, incremental development simplifies system complexity and preserves the required behavior of the system in the abstract model as well as in the correctly refined models.

In this paper, we propose a methodology that reveals the issues involved in the formal modeling and verification of self-adaptive autonomous systems using *correct by construction approach*. Moreover, we report our experience for developing the Automatic Rover Protection (ARP), and provide a set of guidelines for tackling the different issues to avoid collision by preserving the local and global properties of a self-adaptive autonomous system. In this work, we use the Event-B [7] modeling language supported by the Rodin IDE [8], with its rich set of tools. In addition, we also use ProB [9] model checker to analyze and validate the developed models.

The remainder of this paper is organized as follows. Section II presents related work. Section III presents preliminary information for ARP and the Event-B modeling language. The ARP system requirements are presented in Section IV. Section V explores an incremental proof-based formal development of the ARP. Brief discussion is provided in Section VI. Section VI concludes the paper along with future work.

II. Related Work

An autonomous self-adaptive software system modifies its functional behavior in order to adapt new changes within the system or its executional environment [1]. In fact, the system

requirements of such systems guarantee to adapt new changes and to have a safe behavior in case of adverse scenarios. Cheng et al. [1], [10] proposed a research road map for software engineering of self-adaptive systems that has covered four important areas, such as requirements, modeling, engineering and assurances in the development of autonomous self-adaptive systems. The formal development of self-adaptive systems requires the integration of several formal techniques and methods borrowed from autonomic computing, distributed systems, hybrid and self-$\star$ systems and real-time systems. Such an integration remains a challenge for formal system design. An overview on autonomic computing presented in [11], which describes computing paradigm, system applications, associated research issues and challenges.

An autonomous self-adaptive software system modifies its functional behavior in order to adapt new changes within the system or its executional environment [1]. Most of these systems are strongly connected to their operating environment. Such systems need either to adapt to new changes to ensure the functional correctness or to react according to changing environment. System reconfiguration or substitution is a key element to implement such kinds of systems that is proposed by several researchers. In [12], π-calculus and process algebra are used for system modeling, including system reconfiguration, by exploiting behavioral matching based on bi-simulation. An Event-B approach was also proposed in [13]. An extended transaction model is developed to ensure behavior consistency during reconfiguration of distributed systems [14]. The B-method is used for validating dynamic re-configuration of the component-based distributed systems using proofs techniques for consistency checking and temporal requirements [15]. Dynamic reconfiguration allows a system to stay in a stable state using self-configuration and self-healing techniques. Tarasyuk et al. [16] proposed a formal approach for developing dynamic reconfigurable system in Event-B based on probabilistic verification technique to guarantee that system will discover possible reconfiguration strategy to meet service requirements despite failures of its vital components. Model checking of timed automata has been used by [17] to model and study the robustness of self-adaptive decentralized systems. Applying formal methods to self-$\star$ systems originates from the needs of understanding how these systems behave and how they meet their specifications. A self-$\star$ system relies on *emergent behaviors*, resulting from interactions between components of the system [18]. Traditionally, the correctness of self-$\star$ and autonomous systems is validated through the simulation and testing [19]. However, simulation and testing are not sufficient to cover the whole set of possible states of a system. Andriamiarina et al. [**?**] proposed to use correct by construction approach for developing the self-$\star$ systems. Smith et al. [18] have applied the stepwise refinement using Z to study a case of self-reconfiguration, where a set of autonomous robotic agents is able to assemble and to reach a global shape.

Self-adaptative systems allow to optimize their performance under changing environment, and the systems heal themselves when any components fail. Feedback control loops have been identified as crucial elements in realizing the self-adaptation of software system. Kephart et al. [20] presented an approach to organise a control loop in self-adaptive systems using four components: Monitor, Analyze, Plan, and Execute. Weyns et al. [21] proposed a simple notation for describing multiple interacting MAPE loops and a set of patterns, which can provide a generic solution for recurring design problems.

In [22], [23], the development of a hybrid system is proposed using the correct by construction approach, where first, it specifies the discrete model and then refines each event by introducing the continuous elements. It includes the use of a "now" variable, a "click" event that jumps in time to the next instant where an event can be triggered and simulated real numbers. Banach et al. [24] proposed Hybrid Event-B that is an extension of Event-B, which contains pliant events to model continuous behavior by using differential equations during system modeling. It should be noted that existing work [22], [23], [24] do not address the problem of autonomy and systems adaptation. However, in our work, we show refinement based modeling approach to handle autonomous systems, which allow to adapt new changes in the execution environments. It important to know that in this work, our system always performs the possible operations in order to preserve the global safe behavior.

The previous work reported above do not cover the whole characteristics of self-adaptive hybrid systems. Some of them address adaptability and autonomy in discrete cases (e.g. web services) and other address the hybrid case (e.g. smart grids) but do not cover the adaptability properties. None of the studied approach integrates the whole characteristics in formal developments of self-adaptive hybrid systems.

III. Preliminaries

A. Requirements for formal modeling of self-adaptive hybrid systems

Self-adaptive hybrid systems integrate several interacting agents being either software, hardware or hybrid components. They shall fulfill requirements issued from both self-adaptive systems and hybrid systems. We have identified the following topics shall be addressed while developing such systems.

- *Adaptability.* Agents composing the system shall adapt their behaviors to the changes and/or commands issued from the environment. They may be reconfigured at runtime (by restoring their current state) or at static time (by rebooting them and restoring the initial state).

- *Centralized/decentralized control.* Agents may have an autonomous behavior in case of a decentralized control i.e. the observation of the local state of the agent and its environment is enough to control and to ensure the safety of the agent, or be controlled by a central entity in case of centralized control i.e. the agent requires the knowledge of the global state of the system and its environment and reacts to commands of the central controller.

- *Controllability.* Each agent is equipped with a controller, reacting to commands, that maintains the agent in a safety

envelope. When this agent is hybrid system, control theory is used in order to define this controller.

- *Domain modeling.* The definition of controllers, requires to model the behavior of the agent. In the particular case of a hybrid agent, the physics of the system (e.g. Kinematics, mechanics, energy, etc.) shall be modeled. Domain specific models need to be defined.

- *Formal modeling.* To support the formal development of self-adaptive hybrid systems, a formal modeling framework supporting modeling both hybrid systems and autonomous systems. Particularly, modeling controllers which rely on continuous theories shall be enabled by the selected formal modeling framework. Moreover, scalability of the method is a key-point to consider.

In the remainder of this paper, we present a stepwise formal development of a self-adaptive hybrid system relying of the Event-B method. We show that the requirements introduced above are handled by this method.

B. The modeling Framework

Event-B [7] is a formal modeling notation, in which the event-driven approach is borrowed from the B-method [25]. Event-B supports the *correct by construction* approach to design an abstract model and a series of refined models for developing any large and complex system.

1) Event-B models: Contexts and Machines: The Event-B language has two main components, *context* and *machine* (see table I), to characterise the systems. A *context* describes the static structure of a system using *carrier sets* s, *constants* c, *axioms* $A(s,c)$ and *theorems* $T_c(s,c)$, and a *machine* describes the dynamic structure of a system using *variables* v, *invariants* $I(s,c,v)$, *theorems* $T_m(s,c,v)$, *variants* $V(s,c,v)$ and *events* evt. Table I shows a formal organization of a model, in which various clauses (i.e. VARIABLES, EVENTS) are used to introduce the required modeling components for specifying the given system requirements. For instance, the clause VARIABLES represents the state and the clause EVENTS represents the transitions (defined by a Before-After predicate (BA)) of a system. A list of events can be used to model possible system behavior to modify the state variables by providing appropriate *guards* in a *machine*. A model also contains INVARIANTS and THEOREMS clauses to represent its relevant properties to check the correctness of the formalized behavior. A VARIANT clause can be used to introduce convergence properties in a machine. Moreover, the terms like *refines, extends*, and *sees* are mainly used to describe the relation between components of Event-B models.

a) Refinement of Event-B models: The refinement, introduced by the REFINES clause (see TABLE I), decomposes a model (thus a transition system) into another transition system containing more design decisions thus moving from an abstract level to a less abstract one. The refinement allows us to model a system gradually by introducing safety properties at various refinement levels. New variables and new events may be introduced in a new refinement level. These refinements preserve the relation between the refining model and its corresponding refined concrete model, while introducing new events and variables to specify more concrete behavior of the system. The defined abstract and concrete state variables are linked by introducing the *gluing invariants*.

CONTEXT *ctxt_id_2*	**MACHINE** *machine_id_2*
EXTENDS *ctxt_id_1*	**REFINES** *machine_id_1*
SETS s	**SEES** *ctxt_id_2*
CONSTANTS c	**VARIABLES** v
AXIOMS $A(s,c)$	**INVARIANTS** $I(s,c,v)$
THEOREMS $T_c(s,c)$	**THEOREMS** $T_m(s,c,v)$
END	**VARIANT** $V(s,c,v)$
	$V(s,c,v)$
	Event evt = **any** x
	where $G(s,c,v,x)$
	then $v :\mid BA(s,c,v,x,v')$
	end
	END

TABLE I
MODEL STRUCTURE

Theorems	$A(s,c) \Rightarrow T_c(s,c)$
	$A(s,c) \wedge I(s,c,v) \Rightarrow T_m(s,c,v)$
Invariant preservation	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \wedge BA(s,c,v,x,v') \Rightarrow I(s,c,v')$
Event feasibility	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \Rightarrow \exists v'.BA(s,c,v,x,v')$
Variant progress	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \wedge BA(s,c,v,x,v') \Rightarrow V(s,c,v') < V(s,c,v)$

TABLE II
PROOF OBLIGATIONS

b) Checking Event-B correctness: Proof obligations: To verify the correctness of a machine and of refinement, we need to discharge the proof obligations generated for a model and for its refinements. The main proof obligations associated to an Event-B model are listed in Table II, in which the prime notation is used to denote the value of a variable after an event is triggered. These PO require to demonstrate that the theorems hold, each event preserves the invariant (inductive), each event can be triggered (feasibility) and if a variant is declared, it shall decrease.

Regarding refinement, two more relevant proof obligations need to be discharged. First, the simulation PO to show that the new modified action in the refined event is not contradictory to the abstract action and the concrete event simulates the corresponding abstract event. Second, in the refined events, we can strengthen the abstract guards to specify more concrete conditions.

More details on proof obligations can be found in [7].

c) The Rodin platform: The Rodin [26], [27] Platform provides rich tool support for model development using the Event-B language. It includes project management, model development, automatic PO generation, proof assistance, model checking, animation and automatic code generation. Once an Event-B model is modelled and syntactically checked in the Rodin Platform, then a set of proof obligations is generated with the help of the Rodin tools. The integrated Rodin tool attempts to prove automatically the generated proof obligations. To discharge these remaining proof obligations, we require human interaction to assist prover by simplifying the complex expressions and predicates.

d) The Theory Plug-in.: A recent development of the Event-B language allows to extend it with theories [**?**] similar to algebraic specifications. In the Rodin Platform, this development is provided by the *Theory* plug-in [**?**]. We are interested to formalize and analyze the system substitution mechanism related to hybrid systems, in which we need to use the `REAL` datatype to define states variables. Therefore, we rely on the theory *Real*, written by Abrial and Butler[1]. It provides a dense mathematical `REAL` datatype with arithmetic operators, an axiomatic semantics and proof rules.

IV. A CASE STUDY: AUTOMATIC ROVER PROTECTION

A. Informal Description

The basic informal system requirements for ARP is provided by Institut de Recherche Technologique of Toulouse (IRT Saint-Exupéry) in the project of INGEQUIP. In fact, we use this case study to full-fill their goal related to formal modeling to describe the system requirement precisely for identifying any inconsistency. Automatic Rover Protection (ARP) is a function for collision avoidance for TwlRTee (a three wheel rover). ARP is an autonomous system that is a collection of small rovers and these rovers are supervised by a unique supervision station. In general, each rover moves autonomously on a predefined tracks according to the given safety constraints by applying required speed and brake. The main objective of ARP is to provide safe operations among autonomous rovers using a set of constraints to prevent from collision. The unique supervision station supervises regularly each and every tasks of each rover, including position and speed. Moreover, it also maintains a global information of overall system that can be used by any other rovers, and every rover has partial view of the state of the other rovers. In addition, this supervision station can also override any task that can be performed by any rover locally. For example, it can perform stop or brake event on any rover in case of any emergency or any technical difficultly.

B. Requirements and Assumptions

This section provides informal ARP requirements, including required definitions and assumptions to understand the main peculiarity of the system.

1) Definitions:

- **DEF-D1: Warning Area.** Area on which a rover takes an emergency action in presence of any other rover.
- **DEF-D2: Caution Area.** The smallest area on which a rover can stop without reaching at the area boundary.
- **DEF-D3: Info Area.** Area on which a rover can see other rovers without taking any action.

2) Requirements:

- **SAF-R1: Separate rovers in the time and space domains.** All active rovers shall separate in both time and space domains. If any two rovers are not separated then it is known as a *conflict* situation;
  - ***SAF-R1.1: Separate rovers in the time domain.*** The ARP shall detect the temporal conflict situation and react instantly whenever any two active rovers become temporarily too close.
  - ***SAF-R1.2: Separate rovers in the space domain.*** The ARP shall detect the spatial conflict situation and react instantly whenever any two active rovers become spatially too close.
- **FUN-R2: Signal conflict.** An active rover shall detect any conflict situation at most within 100 ms, and the detected conflict must be valid until it disappears.
- **FUN-R3: Provide position data.** Every rover shall broadcast its position, identifier and the required bounded warning area to all other rovers, including supervisor.
- **FUN-R4: Rover priority.** Rover priority shall be computed by the given function f that is a bijective function from the identifiers to the integers.
- **REQ-R5: Avoid conflicts.** In case of a detected conflict situation, a rover shall (i) determine the resolution action, and (ii) apply the resolution action.
  - ***FUN-R5.1: Determine the resolution action.*** The conflict resolution action shall always be taken towards the satisfaction of SAF-R1.1 and SAF-R1.2.
  - ***FUN-R5.2: Apply "STOP" conflict resolution action.*** If the conflict resolution action is "STOP" then the rover shall apply maximum breaking force until it stops.
  - ***FUN-R5.3: Apply "BRAKE" conflict resolution action.*** Two cases shall be possible when "BRAKE" resolution action is triggered on any active rover:
    1) If an active rover has not the priority over the other rovers then the breaking power shall be applied to this rover to ensure that it leaves the warning areas of the other active rovers.
    2) If an active rover has the priority over the other rovers then it does nothing.
- **FUN-R6: Supervisor-initiated action.** A rover performs any collision avoidance action requested by the supervisor: "STOP" and "BRAKE". If a "BRAKE" action is initiated by the supervisor, a supervisor defined deceleration is applied while the "BRAKE" action is active.
- **FUN-R7: Resolution of conflicting actions.** If two actions shall be requested concurrently, i.e. two requests, emitted locally or remotely must be received simultaneously by the rover, or one action must be requested while another is ongoing then,
  1) "STOP" action is taken over "BRAKE" action.
  2) else the supervisor initiated action must be taken over the autonomously taken action.
- **FUN-R8: Rover data when leaving the set of "known rovers".** When a rover leaves the set of "known rovers" then its last position must be accessible to all other rovers.
- **FUN-R9: Recovery from a "STOP" action.** When a "STOP" action shall be performed, the rover must wait for a supervisor action to resume the mission.

[1] http://wiki.event-b.org/index.php/Theory_Plug-in\#Standard_Library

3) Environment Assumptions:

- **ENV-E1: Breaking capability.** A rover shall be able to use maximum force to stop the rover without slipping according to the newton principle.
- **ENV-E2: Bounded speed.** A rover shall not exceed the maximal speed of the edge on which it shall move. It must be ensured by the control function of the rover.
- **HYP-E3: "Off-track" rover management.** When a rover becomes "off-track", it performs the STOP action.
- **HYP-E4: "Active rovers" set management.** All the active rovers must be controlled by the supervisor. For example, adding/removing active rovers must be performed by the supervisor. This information must be broadcasted instantly to all other active rovers.
- **ENV-E5.** Active rovers always move on horizontal plane.

C. Refinement Strategy

In our development, we consider that each rover is self-adaptive and autonomous behaves according to changing environment. It can also be controlled by the supervisor in case of emergency situation. We use a correct by construction approach to design the whole ARP system, in which the first abstract model describes controller operating modes using mode automata to model the possible changing states/modes of each rover. The first refinement introduces space segregation and position of the rovers. In particular, this refinement models a set of spatial ranges and positions for each rover abstractly, including the required safety properties, such as *warning area* is a subset of *caution area*, and *caution area* is a subset of *info area*. The next four refinement levels are used to model ARP controller using physical properties (i.e.such as mass, brake, speed), domain modeling, dynamic controller and clock. These refinements provide possible dynamic behavior of the rovers, in which it cover complex calculations of physical equations of required spatial range according to the given speed, mass and brake of a rover in order to stop safely within the given time interval. The last two refinements are used to model conflict detection and conflict resolution of rovers, in which time domain separation and space domain separation are used to detect conflicts among the rovers and priority order is defined for conflict resolution.

The next sections give more details on each refinement step. For each refinement, we describe the considered requirements and required safety properties.

V. Formal Development of ARP

We describe stepwise formal development of ARP in abstract model and a sequence of refined models based on informal requirements defined in section IV-B.

A. Abstract Model: Controller Operating Modes

An abstract model of ARP specifies only possible changing modes/states of rovers. Fig. 1 depicts possible operating modes of rovers, in which total six modes are considered (see HYP-R4, REQ-R6, REQ-R7, REQ-R9) . These six modes are *Start*, *Move*, *Stop*, *Brake*, *Remove* and *Conflict*. In order to start the

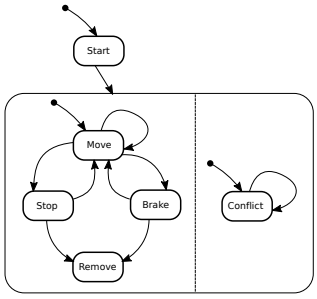

Fig. 1. Operating modes of Rovers

formalization process, we need to define some static properties of the system using Event-B context. The first context declares a set of rovers that is defined by R.

Abstract model of ARP only contains operating modes and possible transition among the modes. To model the dynamic behavior, we declare a list of variables using invariants ($inv1$-$inv4$). These variable are: *r* a set of active rovers; *mr* a set of moving rovers; *rr* a set of removed rovers; *ir* a set of initial rovers that join the set of active rovers; *cr* a set of conflict rovers; *sr* a set of stopped rovers; and *br* a set of braking rovers. In order to define safe behavior, we also define a set of invariants and theorems. The first safety property ($inv8$) states that the total number of rovers from different sets must be equivalent to the set of active rovers. The second safety property ($inv9$) states that no active rover belongs to more than one set at any time.

$inv1 : r \subseteq R \wedge mr \subseteq R \wedge rr \subseteq R$
$inv2 : ir \subseteq R \wedge cr \subseteq R$
$inv3 : sr \subseteq R \wedge br \subseteq R$
$inv8 : ir \cup mr \cup rr \cup sr \cup br \cup cr = r$
$inv9 : ir \cap mr \cap rr \cap sr \cap br \cap cr = \varnothing$
...

EVENT Adding_Rover
ANY x
WHERE
grd1 : $x \in R \setminus r$
THEN
act1 : $ir := ir \cup \{x\}$
act2 : $r := r \cup \{x\}$
END

At this stage, the system describes only discrete functional behavior for changing among various modes without considering any specific requirements. We introduce twelve new events for specifying the mode changing activities in terms of changing states covering the given states of the active rovers. These events include a guard related to the current mode of a rover, and the action changes the modes of the rover. Specially, we show three main events *Adding_Rover*, *Move_to_Move* and *RandomConflict*, which are used to show the mode operation of adding rover, moving rover and to introduce conflict situation randomly.

EVENT Move_to_Move
ANY x
WHERE
grd1 : $x \in mr$
THEN
act1 : $skip$
END

EVENT RandomConflict
ANY x
WHERE
grd1 : $x \in r \setminus cr$
grd2 : $x \notin rr$
THEN
act1 : $cr := cr \cup \{x\}$
act2 : $mr := mr \setminus \{x\}$
act3 $ir := ir \setminus \{x\}$
act4 $sr := sr \setminus \{x\}$
act5 $br := br \setminus \{x\}$
END

An event *Adding_Rover* allows to add a new rover in the set of active rovers (r). The guard of this event shows that x is a

member of R and it does not belong to the set of active rovers r. The actions of this event show that this new rover x is added to the set of active rovers r and the set of initial rovers ir. Another event *Move_to_Move* is formalized in similar fashion. The guard of this event states that a rover x belongs to the set of moving rovers mr and the actions are empty defined as $skip$. To model the conflict behavior at abstract level, we define an event *RandomConflict*. The guards of this event state that a rover x belongs to the set of active rovers, but it does not belong to the set of removed rovers. The actions of this event state that the rover x is added to the set of conflict rovers and it is removed from the other remaining sets.The other events are formalized in a similar way according to Fig. 1.

Remark: In the following sections, we summarize each refinement steps of ARP development, in which from first refinement to sixth refinement only cover *Move_to_Move* development, and the last refinement cover the development of *RandomConflict*. For the sake of simplicity, we omit the detailed formalization and proof details. Our complete formal development of this ARP available on website[2].

B. First Refinement: Space Segregation and Position

This refinement introduces the different types of zone areas (*info area*, *caution area*, *warning area* and *no area*) abstractly for each rover to detect any collision (see DEF-D1, DEF-D2 and DEF-D3, REQ-R3, REQ-R8). A new constant *ZoneSet* is defined in a new *context* for specifying the desired zone requirements for managing the various activities, such as moving, braking, conflicting, removing and stopping, of the rovers according to the rover's speed and rover's position. We also introduce the parameterized set $VALUE$ representing the values of the numbers (being either integers or real values. $VALUE^+$ the restriction of $VALUE$ to the positive numbers.

$$axm1 : ZoneSet \subseteq VALUE$$

A set of variables is declared to define different types of zone definitions using relations for each rover. These variables ($inv1$ - $inv2$) are: ia, ca, wa and na to define *info*, *caution*, *warning* and *no* areas respectively. A new variable *Pos* defines as a function to map rovers r to $ZoneSet$ in $inv3$.

$$\begin{array}{l} inv1 : ia \in r \leftrightarrow ZoneSet \land ca \in r \leftrightarrow ZoneSet \\ inv2 : wa \in r \leftrightarrow ZoneSet \land na \in r \leftrightarrow ZoneSet \\ inv3 : Pos \in r \rightarrow ZoneSet \\ inv4 : \forall x \cdot x \in r \land x \notin rr \Rightarrow Pos(x) \in ran(\{x\} \lhd wa) \\ inv5 : \forall x \cdot x \in r \land x \notin rr \Rightarrow ran(\{x\} \lhd wa) \subseteq ran(\{x\} \lhd ca) \\ inv6 : \forall x \cdot x \in r \land x \notin rr \Rightarrow ran(\{x\} \lhd ca) \subseteq ran(\{x\} \lhd ia) \end{array}$$

A list of safety properties is introduced using invariants to guarantee the safe behavior of the described system. The first safety property ($inv4$) states that the position of every active rover (except removed rovers) always belongs to the warning area wa. The last two safety properties (invariants $inv5$ and $inv6$) state that for each active rover (except removed rovers) the warning area wa is a subset of caution area ca, and the caution area ca is a subset of info area ia, respectively.

In this refinement, we introduce several new guards and actions to specify the desired behavior of rovers according to the current position and various zone definitions for every rover. There are twelve events in total where eleven events refine those of the abstract model. Such as, an event *Move_to_Move* also refines the abstract event *Move_to_Move* by adding several new guards and actions. This refined event contains several state variables, defined in $grd2$ - $grd4$, related to different zones and position. The next guard ($grd5$) states that there is no rover in the warning area wa and caution area ca of rover x. The last guard states that the declared warning area wa_zs is a subset of the declared caution area ca_zs and the declared caution area ca_zs is a subset of the declared info area ia_zs. A set of new actions($act1$-$act4$ is introduced. These actions update the global set of info area ia, caution area ca, warning area wa and no area na with new info area ia_zs, new caution area ca_zs, new warning area wa_zs and new no area na_zs for the rover x, respectively. Finally, the last action ($act5$) updates the position set Pos with the current new position for the rover x.

EVENT Move_to_Move refines Move_to_Move
ANY x, ia_zs, wa_zs, ca_zs, na_zs, new_pos
WHERE
grd1 : $x \in mr$
grd2 : $ia\_zs \subseteq ZoneSet \land ca\_zs \subseteq ZoneSet$
grd3 : $wa\_zs \subseteq ZoneSet \land na\_zs \subseteq ZoneSet$
grd4 : $new\_pos \in wa\_zs$
grd5 : $\forall y \cdot y \in r \land y \neq x \Rightarrow Pos(y) \notin (wa\_zs \cup ca\_zs)$
grd6 : $wa\_zs \subseteq ca\_zs \land ca\_zs \subseteq ia\_zs$
THEN
act1 : $ia := ia \Leftarrow \{i \mapsto j | j \in ia\_zs \land i = x\}$
act2 : $ca := ca \Leftarrow \{i \mapsto j | j \in ca\_zs \land i = x\}$
act3 : $wa := wa \Leftarrow \{i \mapsto j | j \in wa\_zs \land i = x\}$
act4 : $na := na \Leftarrow \{i \mapsto j | j \in na\_zs \land i = x\}$
act5 : $Pos := Pos \Leftarrow \{i \mapsto j | i = x \land j = new\_pos\}$
END

Similarly, another event *Adding_Rover* refines the abstract event *Adding_Rover* by adding new guards and new actions. The refined event *Adding_Rover* has the same guard as the *Move_to_Move* event. Similarly, a set of new actions($act3$-$act6$) associate to the new rover info ia_zs, caution ca_zs, warning wa_zs and no na_zs areas for the new added rover x into the global set of info area ia, caution area ca, warning area wa and no area na, respectively. $act9$ adds the current new position of the rover x into the position set Pos.

EVENT Adding_Rover
ANY x, ia_zs, wa_zs, ca_zs, na_zs, new_pos
WHERE
. . .
THEN
act1 : $ir := ir \cup \{x\}$
act2 : $r := r \cup \{x\}$
act3 : $ia := ia \cup \{i \mapsto j | j \in ia\_zs \land i = x\}$
act4 : $ca := ca \cup \{i \mapsto j | j \in ca\_zs \land i = x\}$
act5 : $wa := wa \cup \{i \mapsto j | j \in wa\_zs \land i = x\}$
act6 : $na := na \cup \{i \mapsto j | j \in na\_zs \land i = x\}$
act7 : $Pos := Pos \cup \{x \mapsto new\_pos\}$
END

Remark: Due to limited space and similar kind of refinement, we omit the described other refining events.

C. Second Refinement: Physical Quantises of ARP

a) Domain modeling: This refinement introduces the physical properties of the rovers, such as mass and speed, used

[2]http://singh.perso.enseeiht.fr/Conference/ICECCS2016/ARPModels.zip

[2]Used symbols in Event-B model: Domain restriction ($\lhd$); Range (ran); Overriding ($\Leftarrow$).

to calculate the required braking force to stop safely any rover (see REQ-R3). We define a set of rover IDs that is defined by $IdSet$. In addition, we declare two constants *Id* and *mass* in axioms ($axm1$ and $axm2$). These constants are used to define unique identification and mass for each rover. The last axiom ($axm3$) is a static property to show that every rover has a unique id.

$axm1 : Id \in R \rightarrow IdSet$
$axm2 : mass \in R \rightarrow VALUE^+$
$axm3 : \forall r1, r2 \cdot r1 \in R \land r2 \in R \land r1 \neq r2 \Rightarrow Id(r1) \neq Id(r2)$

b) Handling physical information: In this refinement, we introduce a variable *speed* ($inv1$). It is defined as a total function to specify the speed for each rover. A set of safety properties state that the current speed of the set of stopped rovers (sr) must be equal 0 in $inv2$; the current speed of the set of braking rovers (br) must be greater then or equal to 0 in $inv3$, the current speed of the set of initial rovers (ir) must be equal to 0 in $inv3$; and the current speed of the set of moving rovers (mr) must be greater than 0 in $inv5$, respectively.

$inv1 : speed \in r \rightarrow VALUE$
$inv2 : \forall x \cdot x \in sr \Rightarrow speed(x) = 0$
$inv3 : \forall x \cdot x \in br \Rightarrow speed(x) \geq 0$
$inv4 : \forall x \cdot x \in ir \Rightarrow speed(x) = 0$
$inv5 : \forall x \cdot x \in mr \Rightarrow speed(x) > 0$

In this refinement, we introduce an extra guard ($grd7$) and an extra action ($act6$) in the refined event *Move_to_Move*. The new introduced guard is used to declare a new variable v of integer type and to ensure that the current speed of the rover x is greater than 0 and the new selected value of the speed v is also higher than 0. The new added action ($act6$) is used to update the value of speed with new selected speed variable v.

EVENT Move_to_Move Refines Move_to_Move
ANY x, ia_zs, wa_zs, ca_zs, na_zs, new_pos, v
WHERE
. . .
grd7 : $v \in \mathbb{Z} \land speed(x) > 0 \land v > 0$
THEN
. . .
act6 $speed(x) := v$
END

D. Third Refinement: Dynamic Controller

Here, the calculation for braking force to stop safely a moving rover is introduced. The abstract definition of area zones are also refined (calculated) according to the applied maximum braking force, current speed and actual mass for each rover (see HYP-R1).

Domain modeling (continued): To define the static properties, the context defining the physics of rovers is enriched. Five new constants (*min_brake*, *max_brake*, *brake_power*, *brake* and *ca_const*) are declared. The first two constants ($axm1$ and $axm2$) define the possible values for minimum brake and maximum brake, respectively. The next constant ($axm3$) is used to define the braking power by providing minimum and maximum ranges. The next constant *brake* is defined as a function to model the brake function for each rover. In $axm5$, we define a constant ca_const as strictly positive that can be used in the model to describe the concrete caution area for each rover. The next two axioms ($axm6$ and $axm7$) are used to define system properties, particularly for brake. These axioms state that the maximum braking power for every rover should not be 0 and for every rover there must exist a maximum braking power. A set of axioms ($axm8$-$axm13$) defines a set of functions for calculating *info area*, *caution area* and *warning area* using mass (m), velocity (v) and brake (b) of a rover.

Remark: The defined axioms correspond to the theory of kinematics and mechanics. This can be seen as an explicit model to formalize domain knowledge [28].

$axm1 : min\_br \in VALUE \land min\_br \geq 0$
$axm2 : max\_br \in VALUE \land max\_br > min\_br$
$axm3 : brake\_power = min\_br \mathinner{..} max\_br$
$axm4 : brake \in R \rightarrow \{brake\_power\}$
$axm5 : ca\_const \in VALUE^+$
$axm6 : \forall r \cdot r \in R \Rightarrow max(brake(r)) \neq 0$
$axm7 : \exists b \cdot b \in VALUE \land (\forall r \cdot r \in R \Rightarrow max(brake(r)) = b)$
$axm8 : ia\_fun \in \mathbb{N} \times VALUE^+ \times VALUE \rightarrow VALUE^+$
$axm9 : \forall m, v, b \cdot m \in VALUE^+ \land v \in VALUE^+ \land b \in VALUE$
$\land \neg b = 0 \Rightarrow ia\_fun(m \mapsto v \mapsto b) = ((2 * m * v * v)/b)$
$axm10 : ca\_fun \in VALUE^+ \times VALUE^+ \times VALUE \rightarrow VALUE^+$
$axm11 : \forall m, v, b \cdot m \in VALUE^+ \land v \in VALUE^+ \land b \in VALUE \land$
$\neg b = 0 \Rightarrow ca\_fun(m \mapsto v \mapsto b) =$
$(((m * v * v)/2 * b) + \ ca\_const)$
$axm12 : wa\_fun \in VALUE^+ \times VALUE^+ \times VALUE \rightarrow VALUE^+$
$axm13 : \forall m, v, b \cdot m \in VALUE^+ \land v \in VALUE^+ \land b \in VALUE$
$\land \neg b = 0 \Rightarrow wa\_fun(m \mapsto v \mapsto b) = ((m * v * v)/b)$

The obtained refinement: It does not declare any new variable, it introduces two safety properties to guarantee the correctness of different areas (*info area*, *caution area*, *warning area* and *no area*) after refining their abstract definition. These safety properties ($inv1$ and $inv2$) state that for each rover, the warning area wa is a subset of the caution area ca, and the caution area ca is a subset of the info area ia.

$inv1 : \forall x \cdot x \in r \land x \notin rr \Rightarrow ran(\{x\} \lhd wa) \subseteq ran(\{x\} \lhd ca)$
$inv2 : \forall x \cdot x \in r \land x \notin rr \Rightarrow ran(\{x\} \lhd ca) \subseteq ran(\{x\} \lhd ia)$

In this refinement, we strengthen the abstract guards by providing more precise information for each defined areas related to *info area*, *caution area* and *warning area*. The strengthening guards $grd2$, $grd3$ and $grd4$, present an actual calculation for different areas according to the mass, current speed and maximum braking power of any rover x.

EVENT Move_to_Move Refines Move_to_Move
ANY x, ia_zs, wa_zs, ca_zs, na_zs, new_pos, v
WHERE
. . .
grd2 : $ia\_zs \subseteq ZoneSet \land (mass(x) \mapsto v \mapsto max(brake(x)))$
$\in dom(ia\_fun) \land ia\_zs = new\_pos \mathinner{..} (new\_pos +$
$ia\_fun(mass(x) \mapsto v \mapsto max(brake(x))))$
grd3 : $ca\_zs \subseteq ZoneSet \land (mass(x) \mapsto v \mapsto max(brake(x)))$
$\in dom(ca\_fun) \land ca\_zs = new\_pos \mathinner{..} (new\_pos +$
$ca\_fun(mass(x) \mapsto v \mapsto max(brake(x))))$
grd4 : $wa\_zs \subseteq ZoneSet \land (mass(x) \mapsto v \mapsto max(brake(x)))$
$\in dom(wa\_fun) \land wa\_zs = new\_pos \mathinner{..} (new\_pos +$
$wa\_fun(mass(x) \mapsto v \mapsto max(brake(x))))$
. . .
THEN
. . .
END

E. Fourth Refinement: Space and Time Domains Separation

This important refinement introduces the time domain separation and space domain separation for each rover in the ARP

framework to avoid any collision and to resolve any conflicts (see REQ-R1, REQ-R5). If any two rovers are not separated then it is known as a conflict situation. The time domain separation states that the ARP shall detect the temporal conflict situation and to react instantly whenever any two active rovers become temporarily too close (see REQ-R1.1), and the space domain separation states that the ARP shall detect the spatial conflict situation and to react instantly whenever any two active rovers become spatially too close (see REQ-R1.2).

To model these requirements we declare a list of constants: *sds_C*, *sds_W*, *sds_I* to define space domain separation and *tds_C*, *tdc_W*, *tds_I* to define time domain separation for $caution$, $warning$ and $info$ areas respectively. Moreover, two additional properties are defined in form of constraints. The first property ($axm2$) states that the space domain separation for caution area must be less than the space domain separation for warning area and the space domain separation for warning area must be less than the space domain separation for info area. The second property ($axm4$) states is similar for time domain separation.

$axm1 : sds\_C \in VALUE \land sds\_W \in VALUE \land sds\_I \in VALUE$
$axm2 : sds\_C < sds\_W \land sds\_W < sds\_I$
$axm3 : tds\_C \in VALUE \land \ldots$
$axm4 : tds\_C < tds\_W \ldots$

$inv1 : sd \in VALUE$
$inv2 : td \in VALUE$

In this refinement, we introduce two variables space domain (sd) in $inv1$ and time domain (td) in $inv2$. They are used to store the values from time and space domains.

In every events we introduce space domain and time domain separations to assign the current value of it according to the *warning area*, *caution area* and *info area* for each rover. In fact, these separations identify any occurred conflict to avoid any collision and to take the required action for safe operations among the rovers.

All the events except *RandomConflict* are refined in this refinement level by adding two actions related to time domain and space domain separations. For example, in event *Move_to_Move*, we add two new actions ($act7$ and $act8$). These actions are defined non-deterministically. They assign a new value of space domain sd that must belong to the space domain of caution, warning or info areas (similarly a new value of time domain td is assigned).

EVENT Move_to_Move Refines Move_to_Move
ANY x, ia_zs, wa_zs, ca_zs, na_zs, new_pos, v
WHERE ...
THEN
...
act7 : $sd :\in \{sds\_C, sds\_W, sds\_I\}$
act8 : $td :\in \{tds\_C, tds\_W, tds\_I\}$
END

F. Fifth Refinement: Clock Introduction

According to the refinement strategy defined in section IV-C, this important refinement introduces a clock to specify the temporal behaviors for modeling the ARP framework. At this level, it is possible to refine the abstract speed and position of the rovers by defining their values inline with time progress.

Time domain modeling: It should be noted that in this model we consider discrete time steps. A total of five constants are declared. The first four constants are: *OBS_mm*, *OBS_mb*, *OBS_mm* and *OBS_csb* for time observation during move to move, move to brake, move to stop and conflict to stop or brake respectively. The next constant $timestep$ and the associated property states that it will be minimum of the time observations (*OBS_mm, OBS_mb, OBS_ms and OBS_csb*). The last constant *Time* defines a set of positive values.

Handling time: A variable *now* is declared to represent the current clock counter in $inv1$, which progresses by the defined constant $timestep$ at every clock tick. In addition, we declare a new variable $Position$ in $inv2$ that is the refinement of the abstract variable $Pos.Position$ is defined as a total function that maps the set of active rovers (except removed rovers) to the pair (time, zone set). Similarly, the next variable $speed\_at$ refines the abstract variable $speed$. Again, it is a total function. It maps the set of active rovers to (time, speed) pair.

$axm1 : OBS\_mm \in VALUE^+ \land OBS\_mb \in VALUE^+ \land$
$axm2 : OBS\_ms \in VALUE^+ \land OBS\_csb \in VALUE^+$
$axm3 : timestep \in VALUE^+ \land$
$\quad timestep = min(\{OBS\_mm, OBS\_ms, OBS\_mb, OBS\_csb\})$
$axm4 : Time = VALUE^+$

To refine the abstract variables, a gluing invariant is introduced to link (glue) abstract and concrete variables. The first gluing invariant $inv4$ rewrites the abstract variable Pos to the concrete variable $Position$, which states that for all active rovers (except removed rovers) the pair of current time and current position are equivalent to the abstract position.

$inv1 : now \in Time$
$inv2 : Position \in r \setminus rr \rightarrow Time \times ZoneSet$
$inv3 : speed\_at \in r \rightarrow Time \times VALUE^+$
$inv4 : \forall x \cdot x \in r \setminus rr \Rightarrow now \mapsto Pos(x) = Position(x)$
$inv5 : \forall x \cdot x \in r \Rightarrow now \mapsto speed(x) = speed\_at(x)$

Similarly, the gluing invariant $inv5$ rewrites the abstract variable $speed$ to the concrete variable $speed\_at$. It states that for all active rovers the pair of the current time and the current speed are equivalent to the abstract speed.

EVENT Move_to_Move Refines Move_to_Move
ANY x, ia_zs, wa_zs, ca_zs, na_zs, new_pos, v, po, sp
WHERE
...
grd8 : $po \in r \setminus rr \rightarrow Time \times ZoneSet$
grd9 : $po = (\{i \mapsto (j \mapsto k) | i = x \land i \in dom(Position) \land j = now + timestep \land k = new\_pos\} \cup \{i \mapsto (j \mapsto k) | i \in r \setminus \{x\} \land i \in dom(Position) \land j = now + timestep \land k = prj2(Position(i))\})$
grd10 : $sp \in r \rightarrow Time \times VALUE^+$
grd11 : $sp = (\{i \mapsto (j \mapsto k) | i = x \land i \in dom(speed\_at) \land j = now + timestep \land k = v\} \cup \{i \mapsto (j \mapsto k) | i \in r \setminus \{x\} \land i \in dom(speed\_at) \land j = now + timestep \land k = prj2(speed\_at(i))\})$
THEN
...
act5 : $Position, speed\_at := po, sp$
act7 : $sd :\in \{sds\_C, sds\_W, sds\_I\}$
act8 : $td :\in \{tds\_C, tds\_W, tds\_I\}$
act9 : $now := now + timestep$
END

Due to introduction of new variables for rewriting the old abstract variables, every event that uses the abstract variables is refined by an event that uses the new variables. For the

case of the *Move_to_Move* event, two new local parametric variables position po and speed sp are used. po is defined as a total function that maps the set of active rovers (except removed rovers) to the pair of (time, zone) set in $grd8$. The next additional guard $grd9$ states that the position set po must be equivalent to the set of pairs of time and position of active rovers (except removed rovers), in which the current position of the rover x is updated to a new position (new_pos) of the rover x. The other active rovers (except removed rovers) keep their own positions at current time (time is increased by a single time step). sp is declared as a total function that maps the set of active rovers to the pair of time and speed. The last additional guard $grd11$ states that the speed set po must be equivalent to the set of pairs of times and speed of active rovers, in which the current speed of the rover x is updated to a new speed v. The other active rovers should have their own speeds at the current time.The other guards of this event are similar to the abstract event. Action $act5$ is modified according to the new defined variables $Position$ and $speed\_at$. It updates the positions and speed of the active rover x. An extra action ($act9$) increases current time now by the time step $timestep$.

G. Sixth Refinement: Conflict Detection

The remaining two refinements correspond to the last step of the refinement strategy defined in section IV-C. A random conflict is triggered. It may occurr, when an active rover becomes temporally or spatially too *close* (in terms of space and time separation as defined in previous refinement of section V-E) to another known rover (REQ-R2, REQ-R8).

$axm1 : S\_Distance \in (Time \times ZoneSet) \times (Time \times ZoneSet) \rightarrow VALUE^+$
$axm2 : S\_Time \in (Time \times PointsSet) \times (Time \times PointsSet) \rightarrow VALUE^+$
$axm3 : PointAt \in R \rightarrow Time \times PointsSet$

Three constants *S_Distance*, *S_Time* and *PointAt* are introduced ($axm1$ - $axm3$). The set *PointsSet* gathers all points on the segments defined in the embedded cartography. The first two total functions (*S_Distance* and *S_Time*) calculate spatial separation and temporal separation and the last one *PointAt* calculates an actual position of the active rovers.

This refinement does not introduce any new variable. Solely the abstract event *RandomConflict* is refined by two new similar events *RandomConflict_TimeDomain* and *RandomConflict_SpaceDomain* for time and space separation violation. *RandomConflict_TimeDomain* introduces a new guard ($grd2$) to assert that current active rover x is temporarily too close to another known active rover and the required time interval between the closed rovers is less than the required interval of time domain separation (td).

EVENT RandomConflict_TimeDomain Refines RandomConflict
ANY x
WHERE
grd1 : $x \in r \setminus cr \wedge x \notin rr$
grd2 : $\exists i \cdot i \in r \setminus rr \wedge S\_Time(PointAt(x) \mapsto PointAt(i)) < td$
THEN
$act1 : cr, mr, ir, sr, br := cr \cup \{x\}, mr \setminus \{x\}, ir \setminus \{x\}, sr \setminus \{x\}, br \setminus \{x\}$
END

H. Seventh Refinement: Conflict Resolution

This is the last refinement that introduces rover's priority that allows to active rovers to perform any action in conflict situation (see REQ-R4, REQ-R2, REQ-R5). For instance, if two rovers are in conflict situation and first rover has higher priority than second rover then the second rover performs the brake or stop action.

Priority policy: We introduce two constants $Priority$ and $HigherPri$ (axioms $axm1$ and $axm2$). The constant $Priority$ is a total function that maps each rover to its priority. $HigherPri$ is also a total function to choose in a pair of rovers, the one with higher priority. In addition, $axm3$, $axm4$ and $axm5$ define the policy associated to the rovers priorities.

$axm1 : Priority \in IdSet \rightarrow \mathbb{N}$
$axm2 : HigherPri \in R \times R \rightarrow R$
$axm3 : \forall r1, r2 \cdot r1 \in R \wedge r2 \in R \wedge r1 \neq r2 \wedge (Priority(Id(r1)) > Priority(Id(r2)) \Leftrightarrow HigherPri(r1 \mapsto r2) = r1)$
$axm4 : \forall r1, r2 \cdot r1 \in R \wedge r2 \in R \wedge r1 \neq r2 \wedge (Priority(Id(r1)) < Priority(Id(r2)) \Leftrightarrow HigherPri(r1 \mapsto r2) = r2)$
$axm5 : \forall r1, r2 \cdot r1 \in R \wedge r2 \in R \wedge r1 \neq r2 \wedge (Priority(Id(r1)) = Priority(Id(r2)) \Leftrightarrow (HigherPri(r1 \mapsto r2) = r1) \vee (HigherPri(r1 \mapsto r2) = r2))$

Refined conflict events: We only refine two events *Conflict_to_Brake* and *Conflict_to_Stop*. In both events, we introduce an extra guard ($grd12$) that states the rover x has lower priority in case of conflict with any other known active rover. The guard $grd12$ is added in the event *Conflict_to_Brake*.

EVENT Conflict_to_Brake Refines Conflict_to_Brake
ANY x, ia_zs, wa_zs, ca_zs, na_zs, new_pos, v, po, sp
WHERE
...
grd12 $\forall i \cdot i \in cr \wedge i \neq x \Rightarrow Priority(Id(x)) < Priority(Id(i))$
THEN
...
END

I. Model Validation and Analysis

Event-B supports *consistency checking* and *refinement checking*. The associated proof obligations (POs) are those of table II. We have conducted our stepwise development using both model checking and proof. Model checking with ProB [9] has been used to validate the behavioral requirements of the developed models. Validation refers to gaining confidence that the developed models are consistent with requirements. ProB supports *automated consistency checking* and *constraint-based checking*. The use of ProB helped us to identify the desired behavior and to guarantee the deadlock freedom of ARP models for different scenarios. As a second step, we have used proof based techniques to definitely prove the correctness of our development. Table III shows the proof statistics of the development in the Rodin IDE. To guarantee the correctness of the system behavior, we incrementally introduce the safety properties in the stepwise refinements. The development resulted in 723 (100%) POs, in which 444 (62%) are proved automatically, and the remaining 279 (38%) are proved interactively using the Rodin prover and SMT solvers.

VI. Conclusion

Automatic Rover Protection (ARP) is a a self-adaptive system implementing a function for collision avoidance for

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	242	153(64%)	89(36%)
First Refinement	108	50(47%)	58(53%)
Second Refinement	70	70(100%)	0(0%)
Third Refinement	111	62(56%)	49(44%)
Fourth Refinement	22	22(100%)	0(0%)
Fifth Refinement	163	82(51%)	81(49%)
Sixth Refinement	2	0(0%)	2(100%)
Seventh Refinement	5	5(100%)	0(0%)
Total	**723**	**444(62%)**	**279(38%)**

TABLE III
PROOF STATISTICS

TwlRTee. The prime goal is to provide safe operations among rovers.

This paper presented a formal development of the ARP self-adaptive system using a correct-by-construction approach and incremental refinement. We described the static and dynamic system properties in form of system requirements using formal notations in the abstract and refined models (8 models in total). As far as we know, this is the first formal model of the ARP to analyze that the functional behavior complies with its safety requirements. We used the Event-B modeling language, together with its associated tools, to develop the proof-based formal model using a refinement technique. Our incremental development reflects not only the many facets of the problem, but also that there is a learning process involved in understanding the problem and its possible solutions. The complete Event-B development is available on our website[3].

Our ultimate goal is to develop a system controller based on our models by refining the current models with correct discretization of continuous behaviors. In particular, the objective is to identify, using control theory, the correct time steps in the developed ARP controller. In addition, we plan to integrate the developed formal models of ARP with physical environment (e.g. cartography) to model the closed-loop system for verifying the desired behavior under relevant safety properties, and be able to guarantee the correctness of the functional behavior of the ARP in chaotic environment. We also intend to implement ARP framework for a hardware platform by generating source code using EB2ALL.

REFERENCES

[1] Cheng et al., "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. LNCS, B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Springer Berlin Heidelberg, 2009, vol. 5525, pp. 1–26.

[2] N. K. Singh, *Using Event-B for Critical Device Software Systems*. Springer-Verlag GmbH, 2013.

[3] M. B. Andriamiarina, D. Méry, and N. K. Singh, "Revisiting snapshot algorithms by refinement-based techniques," *Comput. Sci. Inf. Syst.*, vol. 11, no. 1, pp. 251–270, 2014.

[4] Y. Chen, M. Lawford, H. Wang, and A. Wassyng, "Insulin pump software certification," in *Foundations of Health Information Engineering and Systems*, ser. LNCS, J. Gibbons and W. MacCaull, Eds. Springer Berlin Heidelberg, 2014, vol. 8315, pp. 87–106.

[5] M. Satpathy, S. Ramesh, C. F. Snook, N. K. Singh, and M. J. Butler, "A mixed approach to rigorous development of control designs," in *2013 IEEE International Symposium on Computer-Aided Control System Design, CACSD 2013, Hyderabad, India*, 2013, pp. 7–12.

[6] I. Lee, G. J. Pappas, R. Cleaveland, J. Hatcliff, B. H. Krogh, P. Lee, H. Rubin, and L. Sha, "High-confidence medical device software and systems," *Computer*, vol. 39, no. 4, pp. 33–38, 2006.

[7] J. Abrial, *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[8] Project RODIN, "Rigorous open development environment for complex systems," http://rodin-b-sharp.sourceforge.net/, 2004.

[9] M. Leuschel and M. Butler, *ProB: A Model Checker for B*, ser. LNCS. Springer, 2003, pp. 855–874.

[10] Lemos et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, ser. LNCS, R. Lemos, H. Giese, H. A. Mller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 1–32.

[11] M. Parashar and S. Hariri, "Autonomic computing: An overview," in *Unconventional Programming Paradigms*, ser. Lecture Notes in Computer Science, J.-P. Bantre, P. Fradet, J.-L. Giavitto, and O. Michel, Eds. Springer Berlin Heidelberg, 2005, vol. 3566, pp. 257–269.

[12] A. Bhattacharyya, "Formal modelling and analysis of dynamic reconfiguration of dependable systems," Ph.D. dissertation, Newcastle University School of Computing Science, January 2013.

[13] G. Babin, Y. Ait-Ameur, and M. Pantel, "A generic model for system substitution," in *Trustworthy Cyber Physical Systems Engineering*, A. Romanovsky and F. Ishikawa, Eds. CRC Press Taylor & Francis Group, 2016.

[14] N. De Palma, P. Laumay, and L. Bellissard, "Ensuring dynamic reconfiguration consistency," in *In 6th International Workshop on Component-Oriented Programming (WCOP 2001)*, 2001, pp. 18–24.

[15] A. Lanoix, J. Dormoy, and O. Kouchnarenko, "Combining proof and model-checking to validate reconfigurable architectures," *Electronic Notes in Theoretical Computer Science*, vol. 279, no. 2, pp. 43 – 57, 2011, proceedings of the 8th International FESCA.

[16] I. Pereverzeva, E. Troubitsyna, and L. Laibinis, "A refinement-based approach to developing critical multi-agent systems," *International Journal of Critical Computer-Based Systems*, vol. 4, no. 1, pp. 69–91, Jan 2013.

[17] M. U. Iftikhar and D. Weyns, "A case study on formal verification of self-adaptive behaviors in a decentralized system," vol. 91, 2012, pp. 45–62.

[18] G. Smith and J. W. Sanders, "Formal development of self-organising systems," in *Proceedings of the 6th International Conference on Autonomic and Trusted Computing*, ser. ATC '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 90–104.

[19] M. Puviani, G. D. M. Serugendo, R. Frei, and G. Cabri, "A method fragments approach to methodologies for engineering self-organizing systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 3, pp. 33:1–33:25, Oct. 2012.

[20] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[21] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Goschka, "On patterns for decentralized control in self-adaptive systems." LNCS, 2013, vol. 7475, pp. 76–107.

[22] W. Su, J.-R. Abrial, and H. Zhu, "Formalizing hybrid systems with Event-B and the Rodin platform," *Science of Computer Programming*, vol. 94, Part 2, pp. 164 – 202, 2014.

[23] M. Butler, J.-R. Abrial, and R. Banach, *From Action Systems to Distributed Systems: The Refinement Approach*. Taylor & Francis, 2016, ch. Modelling and Refining Hybrid Systems in Event-B and Rodin.

[24] R. Banach, M. Butler, S. Qin, N. Verma, and H. Zhu, "Core Hybrid Event-B I: Single Hybrid Event-B machines," *Science of Computer Programming*, 2015.

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

[26] M. Jastram, *Rodin User's Handbook*, oct 2013. [Online]. Available: http://handbook.event-b.org

[27] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in Event-B," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 6, pp. 447–466, 2010.

[28] Y. Aït Ameur and D. Méry, "Making explicit domain knowledge in formal system development," *Science of Computer Programming*, vol. 121, pp. 100 – 127, Mar. 2016.

[3] http://singh.perso.enseeiht.fr/Conference/ICECCS2016/ARPModels.zip