An Event-B formalization of KAOS goal refinement patterns

Abderrahman Matoussi, Frédéric Gervais, Régine Laleau

An Event-B formalization of KAOS goal refinement patterns

Abderrahman Matoussi Frédéric Gervais Régine Laleau

January 2010

TR–LACL–2010–1

Laboratoire d'Algorithmique, Complexité et Logique (LACL)
Département d'Informatique
Université Paris 12 – Val de Marne, Faculté des Science et Technologie
61, Avenue du Général de Gaulle, 94010 Créteil cedex France
Tel.: (33)(1) 45 17 16 47, Fax: (33)(1) 45 17 66 01

Laboratory of Algorithmics, Complexity and Logic (LACL)
University Paris 12 (Paris East)

Technical Report **TR–LACL–2010–1**

A. Matoussi, F. Gervais, R. Laleau.
An Event-B formalization of KAOS goal refinement patterns

An Event-B formalization of KAOS goal refinement patterns

Abderrahman Matoussi **Frédéric Gervais** **Régine Laleau**

Laboratory of Algorithmics, Complexity and Logic - University Paris 12 (Paris East),
France
{abderrahman.matoussi,frederic.gervais,laleau}@univ-paris12.fr

Abstract

Goals play an important role in requirements engineering process, and consequently in systems development process. Whereas specifications allow us to answer the question "WHAT the system does", goals allow us to address the "WHY, WHO, WHEN" questions [5]. Up to now, the main software development approaches using formal methods, such as Event-B, begins at the specification level. Our objective is to include requirements analysis within this process, and more precisely the KAOS method, which is a goal-oriented methodology for requirements engineering. The latter allows analysts to build requirements models and to derive requirements documents. Existing work that combine KAOS with formal methods generate a formal specification model from a KAOS requirements model. We aim at expressing KAOS goal models with a formal language (Event-B), hence staying at the same abstraction level. Thus we take advantage from the Event-B method: (i) it is possible to use the method during the whole development process and (ii) we can benefit from the industrial maturity of tools supporting the method. For that purpose, we propose a constructive approach in which Event-B models are built incrementally from KAOS goal models, driven by goal refinement patterns.

Keywords. Requirements engineering, KAOS methodology, Event-B method, KAOS refinement patterns.

1 Introduction

Employing formal methods for complex systems specification is steadily growing from year to year. They have shown their ability to produce such systems for large industrial problems such as Paris metro line 14 [6] or Roissy Val [7] using the B method [1]. With most of formal methods, an initial mathematical model can be refined in multiple steps, until the final refinement contains enough details for an implementation. Most of the time, the initial model is derived from the description, obtained by the requirements analysis. Consequently, the major remaining weakness in the development chain is the gap between textual or semi-formal requirements and the initial formal specification. There is little research on reconciling the requirements phase with the formal specification phase. In fact, the validation of this initial formal specification is very difficult due to the inability to understand formal models (for customers) and to link them with initial requirements (for designers).

Our objective is to combine the requirements and the specification phases by using KAOS and the Event-B method. On one hand, KAOS is a goal-oriented methodology for requirements engineering which allows analysts to build requirements models and to derive requirements documents. On the other hand, Event-B is a model-based formal method which provides language, techniques and tools to support the analysis and design of systems, from the specification to the implementation stages. Existing work [8, 9, 12, 13] that combine KAOS with formal methods generate a formal specification model from a KAOS requirements model. Contrary to these methods that take only a subset of the KAOS models into account, our long-term objective is to express the whole KAOS requirements model with Event-B, in order to support formal reasoning on it. The key idea is to stay at the same abstraction level as KAOS.

In this report, we begin by considering the first stage of KAOS requirements analysis, namely the goals modeling. In that aim, we present: i) an Event-B semantics for a subset of the KAOS goals called ”Achieve” goals, and ii) a formalization in Event-B of the most used refinement patterns of KAOS goals. This report continues our previous works [10, 11] by addressing the Event-B formalization of KAOS pattern with additional studies, results and proofs. The Event-B formalization of the other KAOS models is a work in progress. The remainder of this report is organized as follows. Section 2 overviews the KAOS and the Event-B formal methods that are employed in the proposed approach. Section 3 details our proposed approach that consists to express KAOS goals model with Event-B. Sections 4, 5 and 6 present the Event-B formalization of, respectively, the milestone-driven goal refinement pattern, the AND goal refinement pattern and the OR goal refinement pattern. Section 7 overviews some other KAOS goal refinement patterns. Section 8 illustrates the approach with a case study. Relevant issues and related work are discussed in Section 9 and 10. Finally, Section 11 concludes with an outline of future work.

2 Background

This section briefly describes the two methods that the proposed approach is based on namely the Event-B formal method and the KAOS methodology.

2.1 KAOS method

KAOS (Knowledge Acquisition in autOmated Specification) [4] is a goal-based requirements engineering method. KAOS requires the building of a data model in UML-like notation. A goal defines an objective the system should meet, usually through the cooperation of multiple agents such as devices or humans. KAOS differentiates between goals and domain properties that are descriptive statements about the environment such as physical laws, organizational norms or policies, etc. KAOS is composed of five complementary sub-models related through inter-model consistency rules:

- The central model is the **goal model** which describes the goals of a system and its environment. The core of the goal model consists of a refinement graph showing how higher-level goals are refined (using the concept of refinement patterns [5]) into lower-level ones and, conversely, how lower-level goals contribute to higher-level ones. Higher-level goals are strategic and coarse-grained while lower-level goals are technical and fine-grained (more operational in nature).

- The **object model** defines the objects (agents, entity...) of interest in the application domain.

- The **agent responsibility model** takes care of assigning goals to agents in a realizable way.
- The **operation model** captures the system operations in terms of their individual features and their links to the goal, object and agent models.
- The **behavior model** captures the required behaviors of system agents in terms of temporal sequences of state transitions for the variables they control.

The importance of the goal model derives from the central role played by goals in the requirements engineering (RE) process. For instance, we can derive other models in a systematic way from the goal model such as the object and operation models. Moreover, the goal model enables early forms of RE-specific analysis such as risk analysis, conflict analysis, or evaluation of alternative options.

KAOS provides a catalog of goal patterns that generalize the most common goal configurations. *Achieve Goals* specifies a property that the system will achieve "some time in the future". *Cease Goals* disallow achievement "some time in the future". *Maintain Goals* specifies a property that must hold "at all times in the future". *Avoid Goals* prescribes a property that must not hold "at all times in the future".

Goals in KAOS can be either "AND" or "OR" refined. A goal is AND-refined into subgoals, such that the conjunction of the subgoals is a sufficient condition to achieve the parent goal. The OR-refinement associates a goal to a set of alternative subgoals in which the achievement of the higher-level goal requires the achievement of at least one of its subgoals. KAOS offers a lot of refinement patterns [5] that decompose goals. These patterns can only be used in the context of different tactics defined in KAOS such as *the milestone-driven tactics* which consists in identifying milestone states that must be reached to achieve the target predicate.

KAOS also provides a criterion for stopping the refinement process. If a goal can be assigned to the sole responsibility of an individual agent, there is no need for further goal refinement to occur. Operational goals (goals that are assigned to agents) are the leaves of a goal graph. Each leaf can be either a requirement (if it is assigned to an agent of the system) or an expectation (if it is assigned to an agent in the environment). The reader may refer to [4] for a full description of these notions.

Notice that KAOS provides an optional formal assertion layer for the specification of goals in Real-Time Linear Temporal Logic (RT-LTL). This formalization step [5, 4] allows to check for instance that goal refinements are correct and complete using a theorem prover, formal refinement patterns, or a bounded SAT solver. Even if such checking is important in order for example to detect missing subgoals in incomplete requirements, the use of this kind of logic cannot fill in the gap between requirements and the later phases of development. This is a serious shortcoming since it obliges designers to use another formal method for developing their systems. Consequently, it is difficult to validate specifications with regard to requirements even if they have been expressed with RT-LTL.

2.2 Event-B method

Event-B [3], an evolution of the classical B method [1], is a formal method for modeling discrete systems by refinement. An Event-B model can be described in terms of two basic constructs:

- **The context:** it provides axiomatic properties of Event-B models. It contains the static part of a model such as carrier sets, constants, axioms and theorems. Carrier sets are similar to types but both, carrier sets and constants, can be instantiated. Axioms describe properties of carrier sets and constants. Theorems are derived properties that can be proved from the axioms. Proof obligations associated with contexts are straightforward: the stated theorems must be proved.

- **The machine:** it contains the dynamic part such as variables, invariants, theorems, events and variants. Variables v define the state of a machine. Possible state changes are described by means of events. Each event is composed of a guard $G(t, v)$ and an action $S(t, v)$, where t are local variables the event may contain. The guard states the necessary condition under which an event may occur, and the action describes how the state variables evolve when the event occurs. The correctness of an Event-B model is defined by an invariant property which every state in the system must satisfy. So, every event in the system must be shown to preserve this invariant. In order to verify this requirement, proof obligations have been defined.

It is also important to indicate that the most important feature provided by Event-B is its ability to stepwise refine specifications. Refinement is a process that transforms an abstract and non-deterministic specification into a concrete and deterministic system that preserves the functionality of the original specification. During the refinement, event descriptions are rewritten to take new variables into account. This is performed by strengthening their guards and adding substitutions on the new variables. New events that only assign the new variables may also be introduced. Proof obligations (POs) are generated to ensure the correctness of the refinement with respect to the abstract model. Event-B is supported by several tools, currently in the form a platform called Rodin [28].

3 Expressing Goals in Event-B

3.1 Motivation

In the software development life-cycle, the first phase corresponds to the requirements engineering. It is followed by the specification phase and then, the development stage. The Event-B method has shown that it was very relevant for the last two phases. So, the approach proposed in this report aims at introducing requirements analysis into the Event-B method. Thus, it will be possible to establish formal links between this model and the specification of a system. As we said before, we have chosen KAOS as a goal-oriented RE method because in KAOS the emphasis was more on semi-formal and formal reasoning about behavioral goals for derivation of goal refinements, goal operationalizations, etc, while in i* [20] for example (the other famous goal-oriented RE), the emphasis was more on qualitative reasoning on soft goals. The choice of Event-B is due to its similarity and complementarity with KAOS: (i) both Event-B and KAOS have notions of refinement and constructive approach; (ii) KAOS and Event-B (conversely to the classical B) have the ability to model both the system and its environment. Hence, these points offer the possibility of using Event-B in the early phases of the development of complex systems. This can help the designers in constructing a mathematical simulation of the overall system.

Since goals play an important role in requirements engineering process and provide a bridge linking stakeholder requests to system specification [24], the proposed approach comes down to systematically derive an Event-B representation from a KAOS goal model. Consequently, we show that it is possible to express KAOS goal models with formal method like Event-B by staying at the same abstraction level. However, it is not possible to verify that both models are equivalent since KAOS models are only semi-formal. The Event-B expression of a KAOS goal model allows to give it a precise semantics just as existing translations from UML specifications into B specifications give a formal semantics to class diagrams or state diagrams [18, 19].

3.2 Achieve Patterns

To achieve our objective, we formalize with Event-B the KAOS refinement patterns that analysts use to generate a KAOS goal hierarchy. We think that these formal design patterns or proof-based design patterns will be very useful and explores the fact that the Event-B method provides a framework for developing generic models of systems. In this report, we focus on the most frequently used goal patterns: the Achieve goals. Formalization of the other categories of goal patterns is a work in progress. An Achieve goal prescribes intended behaviors where some target condition must sooner or later hold whenever some other condition holds in the current system state (this state is an arbitrary current one). An Achieve goal in KAOS is denoted as follows: *Achieve[**TargetCondition** From **CurrentCondition**]*. This notation has the following informal temporal pattern where ***CurrentCondition*** prefix is optional (said otherwise, it can be true):

*[if **CurrentCondition** then] sooner-or-later **TargetCondition**.*

3.3 Formalization of KAOS Achieve Goals

If we refer to the concepts of guard and postcondition that exist in Event-B, a KAOS goal can be considered as a postcondition of the system, since it means that a property must be established. The crux of our formalization is to express each KAOS goal as a Event-B event, where the action represents the achievement of the goal. Then, we will use the Event-B refinement relation and additional custom-built proof obligations to derive all the subgoals of the system by means of Event-B events. Let us consider now that the Achieve goal G is refined into two sub-goals G_1 and G_2 as shown in Figure 1.

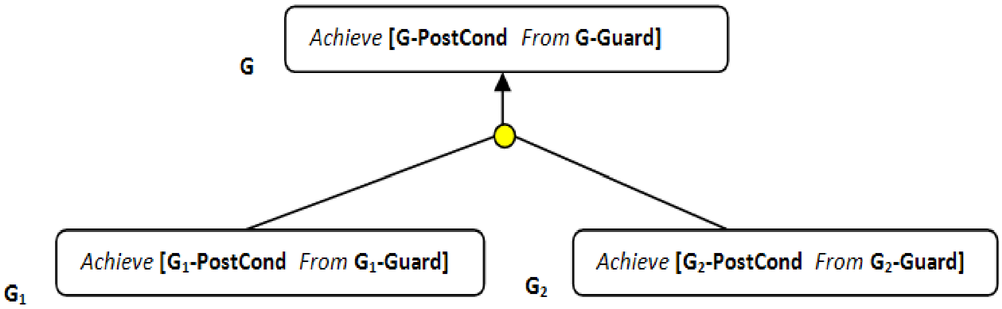

Achieve [G-PostCond From G-Guard]
G
Achieve [G1-PostCond From G1-Guard]
G1
Achieve [G2-PostCond From G2-Guard]
G2

Figure 1: Example of KAOS goal model

Each level i ($i \in [0..n]$) is represented in the hierarchy of the KAOS goal graph as an Event-B model M_i that refines the model M_{i-1} related to the level $i-1$. Moreover, we represent

MACHINE Abstract M_0	MACHINE First M_1
	REFINES Abstract M_0
EvG $\triangleq$	**EvG1** $\triangleq$
when G-$Guard$	**when** G_1-$Guard$
then G-$PostCond$	**then** G_1-$PostCond$
end	**end**
	EvG2 $\triangleq$
	when G_2-$Guard$
	then G_2-$PostCond$
	end
(a) Abstract Model M_0	(b) Refinement Model M_1

Figure 2: Overview of the Event-B representation of the KAOS goal model

each goal as a Event-B event where: (i) the current condition of this goal is considered as the guard; (ii) the **then** part encapsulates the target condition of this goal (see Figure 2).

One may wonder how the temporal characteristic (”sooner or later” keywords) is expressed in our Event-B formalization, as this should imply to specify the concept of time. However, J.R Abrial [2] explains that the time dimension does not need to be explicitly considered at the most abstract levels of a specification, and advocates to use events themselves to express this dimension. Thus, in Event-B, each observable event is so small (atomic) that its execution can be considered to take no time, and then only one event can take place within one unit of time. Consequently, when the unit is large, the corresponding time is very abstract, and when the unit is small, the corresponding time is very concrete. In other words, time is stretched when moving from an abstraction to its refinement. This stretching reveals some ”time details”, some new events. This interpretation suits well to the definition of an Achieve goal. However, if deadlines were associated with such a goal, we would be obliged to introduce new variables to model time. This point will be considered in future work when non-functional goals will be formalized.

In the next sections, we propose an Event-B semantics for each KAOS refinement pattern associated to Achieve goals. Based on the classical set of inference rules from Event-B [3], we have identified the systematic proof obligations for each KAOS goal refinement pattern. In that aim, the following outline will be used for describing each refinement pattern:

1. **Description of the KAOS pattern:** We give a short informal definition of the KAOS goal refinement pattern.

2. **Formal semantics of the pattern:** We propose to give an Event-B semantics to the KAOS goal refinement pattern as follows:
   (a) **Formal definition:** We present the Event-B semantics given to each pattern by constructing some set-theoretic mathematical models based on trace semantics of Event-B developments [3].

   (b) **Proof obligations identification:** We present some informal and formal arguments defining exactly what we have to prove for each KAOS goal refinement pattern.

3. **Synthesis:** We summarize the Event-B formalization of each KAOS goal refinement pattern.

4 Expressing the milestone-driven goal refinement pattern in Event-B.

4.1 Description of the KAOS pattern

The milestone-driven goal refinement pattern [4] refines an Achieve goal by introducing intermediate milestone states $G_1, ..., G_n$ for reaching a state satisfying the target condition (denoted by $G\text{-}PostCond$) from a state satisfying the current condition (denoted by $G\text{-}Guard$) as shown in Figure 3 (with just two sub-goals).

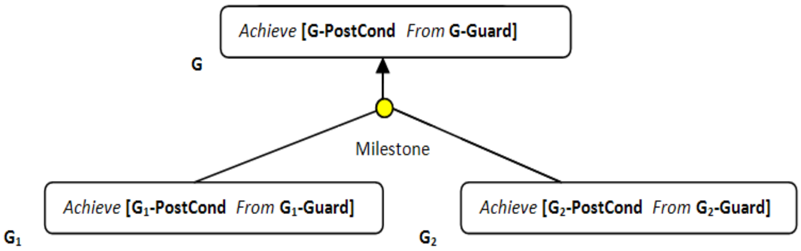

Achieve [G-PostCond From G-Guard]
G
Milestone
Achieve [G₁-PostCond From G₁-Guard]
G₁
Achieve [G₂-PostCond From G₂-Guard]
G₂

Figure 3: Milestone-driven goal refinement pattern

The first sub-goal G_1 is an Achieve goal with the milestone condition as target condition; it states that sooner or later the milestone condition (denoted by $G_1\text{-}PostCond$) must hold if the specific current condition $G_1\text{-}Guard$ (which can be larger than the current condition $G\text{-}Guard$ of the parent goal) holds in the current state. The second sub-goal is an Achieve goal as well; it states that sooner or later the specific target condition $G_2\text{-}PostCond$ (which can be larger than the target condition $G\text{-}PostCond$ of the parent goal) must hold if the specific milestone condition $G_2\text{-}Guard$ (derived from $G_1\text{-}PostCond$) holds in the current state.

4.2 Formal semantics of the pattern

Since the satisfaction of all the KAOS sub-goals (according to a specific order) implies the satisfaction of the parent goal, the abstract event **EvG** is refined by the *sequence* of all the new events (**EvG1**, **EvG2**).

4.2.1 Formal definition

We propose a syntactic extension of the Event-B refinement proof rule in order to provide a way to refine an abstract event by a sequence of new events as follows:

(**EvG1** ; **EvG2**) Refines **EvG**

It is necessary to define the different proof obligations associated to this new refinement semantics. Our idea is to try to characterize more accurately refinement in terms of trace comparisons [3, 23] as illustrated in the following diagram.

$$\begin{tikzcd}
S \arrow[rr, "EvG"] & & S \\
T \arrow[u, "r"'] \arrow[r, "EvG_1"'] & T \arrow[r, "EvG_2"'] & T \arrow[u, "r"]
\end{tikzcd}$$

$$S \triangleq \{v | I(v) \}$$

$$T \triangleq \{w | \exists v \ . \ (I(v) \wedge J(v,w)) \}$$

$$EvG \triangleq \{v \mapsto v' | I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}PostCond(v,v') \}$$

$$EvG1 \triangleq \{w \mapsto t | \exists v \ . \ (I(v) \wedge J(v,w)) \wedge G_1\text{-}Guard(w) \wedge G_1\text{-}PostCond(w,t) \}$$

$$EvG2 \triangleq \{t \mapsto w' | \exists v' \ . \ (I(v') \wedge J(v',w')) \wedge G_2\text{-}Guard(t) \wedge G_2\text{-}PostCond(t,w') \}$$

$$r \triangleq \{w \mapsto v | I(v) \wedge J(v,w)) \}$$

$$r^{-1} \triangleq \{v \mapsto w | I(v) \wedge J(v,w)) \}$$

Figure 4: Set theoretic representation of the discrete models

The Event-B proof obligations must establish the well-known "forward simulation" [23] which is sufficient to guarantee refinement. Indeed, this condition ensures that any trace of the refined model (**EvG1** ; **EvG2**) must also be a trace of the abstraction **EvG** as follows:

$$r^{-1}; EvG1; EvG2 \subseteq EvG; r^{-1}$$

where r a total binary relation from the concrete set T to the abstract set S. r formalizes the gluing invariant between the refined state and the abstract one. In order to translate this condition, it suffices to link S, T, EvG, $EvG1$, $EvG2$ and r with this new formulation. Based on the set theoretic representation of the discrete models [3], each event is defined by means of its guard and before-after predicate as stated in Figure 4.

To well express this refinement semantics, we shall extend the above set theoretic representation (see Figure 4) by expressing the sequence $EvG1; EvG2$ as follows:

$$\begin{aligned} EvG1; EvG2 \triangleq \{w \mapsto w' | \exists v \ . \ I(v) \wedge J(v,w) \wedge \exists t \ . \ G_1\text{-}Guard(w) \wedge G_1\text{-}PostCond(w,t) \wedge \\ G_2\text{-}Guard(t) \wedge G_2\text{-}PostCond(t,w') \} \end{aligned}$$

This definition expresses that we start in a state w where only **EvG1** could be fired; i.e. its guard holds. Once this event is executed (its post-condition became true), we reach immediately another intermediate state t in which the event **EvG2** can be fired. The execution of this last event allow us to reach the state w'. Based on this last definition, the two parts of the forward simulation condition can be stated as follows:

$$\begin{array}{l} r^{-1}; EvG1; EvG2 \triangleq \{v \mapsto w' \mid \exists\, w\ .\ I(v) \wedge J(v,w) \wedge \exists\, t. G_1\text{-}Guard(w) \wedge \\ \qquad\qquad\qquad G_1\text{-}PostCond(w,t) \wedge G_2\text{-}Guard(t) \wedge G_2\text{-}PostCond(t,w')\ \} \end{array}$$

$$EvG; r^{-1} \triangleq \{v \mapsto w' \mid \exists\, v'. I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w')\ \}$$

4.2.2 Proof obligations identification

Based on the above forward simulation condition, we are going to give systematic rules defining exactly what we have to prove for this pattern. In fact, the translation of the forward simulation condition, namely $r^{-1}; EvG1; EvG2 \subseteq EvG; r^{-1}$, comes down to prove the following sequent[1]:

$$\boxed{\begin{array}{l} I(v) \wedge J(v,w) \wedge G_1\text{-}Guard(w) \wedge G_1\text{-}PostCond(w,t) \wedge G_2\text{-}Guard(t) \wedge G_2\text{-}PostCond(t,w') \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'. G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}}$$

Driven by the definition of the milestone refinement pattern, let us suppose that the target condition of the goal G_1 implies the current condition of the goal G_2 as follows:

$$G_1\text{-}PostCond \Rightarrow G_2\text{-}Guard \qquad \textbf{(PO1)}$$

This first proof obligation allows us to simplify the last sequent as follows:

$$\boxed{\begin{array}{l} I(v)\ \wedge\ J(v,w)\ \wedge\ G_1\text{-}Guard(w)\ \wedge\ G_1\text{-}PostCond(w,t)\ \wedge\ G_2\text{-} \\ PostCond(t,w') \\ \vdash \\ G\text{-}Guard(v) \wedge \exists\, v'. G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}} \qquad \text{(I)}$$

We know exactly now which sequent we have to prove. It is the only required condition to verify in order to ensure the consistency of the KAOS milestone-driven goal refinement pattern. This sequent (I) can be split into two sequents ((I.1) and (I.2)) as follows by applying the inference rule *AND-R*[2]:

$$\boxed{\begin{array}{l} I(v) \\ J(v,w) \\ G_1\text{-}Guard(w) \\ G_1\text{-}PostCond(w,t) \\ G_2\text{-}PostCond(t,w') \\ \vdash \\ G\text{-}Guard(v) \end{array}} \qquad \text{(I.1)}$$

[1] The symbol $\vdash$ is named the turnstile. The part situated on the left hand part of the turnstile denotes a finite set of predicates called the hypotheses. The part situated on the right hand side of the turnstile denotes a predicate called the goal [3].

[2] It allows us to simplify conjunctive predicates appearing in the goal of a sequent (see [3]).

Inspired by the definition of the milestone goal refinement pattern given by [4], we ensure that proving the following proof obligation is sufficient to prove the sequent:

$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \qquad \textbf{(PO2)}$$

$$\boxed{\begin{array}{l} I(v) \\ J(v,w) \\ G_1\text{-}Guard(w) \\ G_1\text{-}PostCond(w,t)) \\ G_2\text{-}PostCond(t,w') \\ \vdash \\ \exists\, v'.(G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w')) \end{array}} \tag{I.2}$$

As for the the sequent (I.1), we guarantee that proving the following proof obligation is sufficient to prove the sequent:

$$G_2\text{-}PostCond \Rightarrow G\text{-}PostCond \qquad \textbf{(PO3)}$$

4.3 Synthesis

The milestone-driven goal refinement pattern is expressed in Event-B by considering that the abstract event **EvG** is refined by the sequence of all the new events (**EvG1** and **EvG2**). For that, we have presented the different sufficient proof obligations (not necessary) associated to the syntactic extension of the refinement of Event-B that provides a way to refine an abstract event by a sequence of new events as follows. Of course, these different proof obligations must be in practice considered under an additional hypothesis containing a model invariant I, a gluing invariant J and a collection P of axioms constraining constants and sets.

- *The ordering constraint* **(PO1)** expresses the "milestone" characteristic between the Event-B events.
- *The guard strengthening* **(PO2)** ensures that the concrete guard is stronger than the abstract one. In other words, it is not possible to have the concrete version enabled whereas the abstract one would not. The term "stronger" means that the concrete guard implies the abstract guard.
- *The correct refinement* **(PO3)** ensures that the sequence of concrete events transforms the concrete variables in a way which does not contradict the abstract event.

5 Expressing the AND goal refinement in Event-B

5.1 Description of the KAOS pattern

An Achieve goal G is AND refined into two (or more) sub-goals if the conjunction of the sub-goals is sufficient to establish the satisfaction of the parent goal G as shown in Figure 5 (with just two sub-goals).

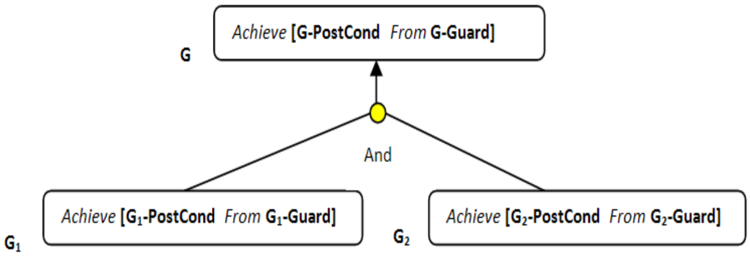

G
Achieve [G-PostCond From G-Guard]
And
G1
Achieve [G1-PostCond From G1-Guard]
G2
Achieve [G2-PostCond From G2-Guard]

Figure 5: AND goal refinement

In Figure 5, the AND-refinement link expresses that the goal G is satisfied by satisfying the goal G_1 and the goal G_2.

Restriction: In KAOS nothing is said on the execution order of the different sub-goals. The formalization in Event-B reveals that this can constitute a serious problem when these sub-goals manipulate shared variables. Let us explain more this problem by supposing that a shared variable x is modified by two sub-goals (G_1 and G_2). G_1 increases x by 3 and G_2 divides x by 2. Hence, it is clear that the execution order of the two sub-goals is important. To completely avoid this problem, a sufficient condition is to force a AND refinement to manipulate only *disjoint set of variables*. This strong solution is quite close to other solutions adopted by a lot of researchers such as J.R Abrial [1] with the parallel behavior concept. If there is a AND refinement with *shared variables*, we propose to transform this form of AND into a "milestone" refinement, and then to explicitly specify the order of modifications on the shared variables.

5.2 Formal semantics of the pattern

As for the milestone-driven tactic, the satisfaction of all the KAOS sub-goals implies the satisfaction of the parent goal. However, the execution of these new events must not necessary follows a specific order. Hence, our idea (inspired from Process Algebra [21]) is that these events (**EvG1** , **EvG2**) are executed in an arbitrary order: either **EVG1;EVG2** or **EVG2;EVG1**. This corresponds to the semantics of the interleave operator in process algebra.

5.2.1 Formal definition

We propose a syntactic extension of the Event-B refinement proof rule in order to refine an abstract event by the interleaving of all the new events as follows:

$$(\mathbf{EvG1}\ |||\ \mathbf{EvG2})\ \text{Refines}\ \mathbf{EvG}$$

As for the milestone-driven tactic, we try to characterize more accurately refinement in terms of trace comparisons as illustrated in the following diagram.

$$\begin{tikzcd}
S \arrow[r, "EvG"] & S \\
T \arrow[u, "r"] \arrow[r, "EvG_1 \quad ||| \quad EvG_2"'] & T \arrow[u, "r"']
\end{tikzcd}$$

We have thus the following "forward simulation" proof to perform in order to ensure that any trace of a refined model must also be a trace of the abstraction:

$$r^{-1}; (EvG1|||EvG2) \subseteq EvG; r^{-1}$$

To well express this new refinement semantics, we shall extend the above set theoretic representation (see Figure 4) by expressing the interleaving $EvG1|||EvG2$ as follows:

$$\begin{array}{l} EvG1|||EvG2 \triangleq \{w \mapsto w' \mid \exists\, v\ .\ I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \Rightarrow G_1\text{-}PostCond(w,w') \wedge G_2\text{-}PostCond(w,w'))\} \end{array}$$

This definition expresses that we start in a state w where both **EvG1** and **EvG2** can be fired; i.e. their guards holds. Hence, we can achieved another state w' iff both events are executed. Based on this last definition, the two parts of the forward simulation condition can be stated as follows:

$$\begin{array}{l} r^{-1}; (EvG1|||EvG2) \triangleq \{v \mapsto w' \mid \exists\, w\ .\ I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \Rightarrow G_1\text{-}PostCond(w,w') \wedge G_2\text{-}PostCond(w,w'))\} \\ EvG; r^{-1} \triangleq \{v \mapsto w' \mid \exists\, v'.I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w')\ \} \end{array}$$

5.2.2 Proof obligations identification

Based on the above forward simulation condition, we are going to give systematic rules defining exactly what we have to prove in order to ensure that a concrete event indeed refines its abstraction. Indeed, the translation of the forward simulation condition, namely $r^{-1}; (EvG1|||EvG2) \subseteq EvG; r^{-1}$, requires to prove the following sequent:

$$\begin{array}{l} I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \Rightarrow G_1\text{-}PostCond(w,w') \wedge G_2\text{-}PostCond(w,w')) \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}$$

This sequent must be simplified using inference rule *IMP-L*[3] as follows:

$$\begin{array}{l} I(v) \wedge J(v,w) \wedge \\ (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ (G_1\text{-}PostCond(w,w') \wedge G_2\text{-}PostCond(w,w')) \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}$$

[3] It allows us to simplify implicative predicates appearing in the hypothesis part of a sequent (see [3]).

The definition of the AND goal refinement pattern given by [4] help us to detect the sufficient proof obligations in order to prove the last sequent; i.e. we ensure that proving these proof obligations is sufficient to prove the sequent:

$G_1\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO1)**
$G_2\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO2)**
$(G_1\text{-}PostCond \wedge G_2\text{-}PostCond) \Rightarrow G\text{-}PostCond$ **(PO3)**

5.3 Synthesis

The AND goal refinement pattern is expressed in Event-B by considering that the abstract event **EvG** is refined by the interleaving of all the new events (**EvG1** and **EvG2**). For that, we have presented the sufficient proof obligations associated to the syntactic extension of the refinement of Event-B that provides a way to refine an abstract event by an interleaving of new events as follows:

- *The guard strengthening* **(PO1, PO2)** ensures that the concrete guard is stronger than the abstract guard of **EvG**. The concrete guard of **EvG1** ||| **EvG2** can be either G_1-*Guard* (if we execute **EvG1** at first) or G_2-*Guard* (if we execute **EvG2** at first).
- *The correct refinement* **(PO3)** ensures that the interleaving of concrete events **EvG1** ||| **EvG2** transforms the concrete variables in a way which does not contradict the abstract event.

Based on the description of all the possible behaviors, we give in **Appendix A** another trace semantics that allows us to discover the above proof obligations. However, the new semantics requires to prove the *associativity condition* if we process more than two events.

6 Expressing the OR goal refinement in Event-B

6.1 Description of the KAOS pattern

An Achieve goal G is OR refined into two sub-goals G_1 and G_2 if only either (not both) of its sub-goals is achieved. A typical OR is shown in Figure 6 (with just two sub-goals).

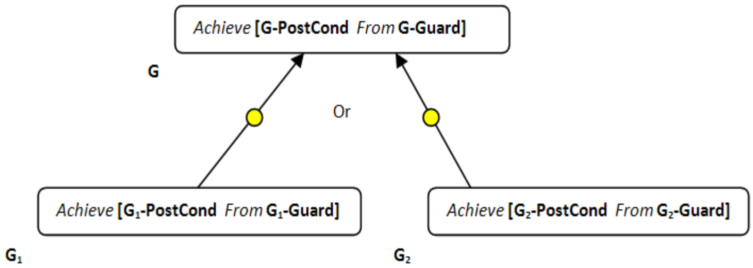

Achieve [G-PostCond From G-Guard]
G
Or
Achieve [G₁-PostCond From G₁-Guard]
G₁
Achieve [G₂-PostCond From G₂-Guard]
G₂

Figure 6: OR goal refinement pattern

In each sub-goal, a sub-goal target condition must be reached in order to reach the target condition *G-PostCond* of the parent goal. Notice that this goal refinement pattern refinement introduces a certain kind of bounded non-determinism that will be resolved further in the implementation phase.

6.2 Formal semantics of the pattern

6.2.1 Formal definition

Since the satisfaction of exactly one KAOS sub-goal implies the satisfaction of the parent goal, we propose to refine the abstract event **EvG** as follows:

$$(\textbf{EvG1}\ \mathit{XOR}\ \textbf{EvG2})\ \text{Refines}\ \textbf{EvG}$$

This exclusive-OR refinement can be see as an inclusive OR refinement with an additional exclusivity characteristic as follows:

$$(\textbf{EvG1}\ \mathit{XOR}\ \textbf{EvG2})\ \text{Refines}\ \textbf{EvG} = \begin{cases} (\textbf{EvG1}\ \mathit{OR}\ \textbf{EvG2})\ \text{Refines}\ \textbf{EvG} \\ \text{Exclusivity characteristic} \end{cases} \qquad \mathit{(OR.def)}$$

As for the other refinement patterns, the first part of the last definition *(OR.def)* can be characterized in terms of trace comparisons as illustrated in the following diagram.

$$\begin{tikzcd} S \arrow[rr, "EvG"] & & S \\ T \arrow[u, "r"'] \arrow[rr, "EvG_1 \quad OR \quad EvG_2"'] & & T \arrow[u, "r"] \end{tikzcd}$$

We have thus the following "forward simulation" proof to perform in order to ensure that any trace of a refined model must also be a trace of the abstraction:

$$r^{-1}; (EvG1\ \text{OR}\ EvG2) \subseteq EvG; r^{-1}$$

To well express this new refinement semantics, we shall extend the above set theoretic representation (see Figure 4) by expressing $EvG1$ OR $EvG2$ as follows:

$$EvG1 \text{ OR } EvG2 \triangleq \{w \mapsto w' \mid \exists\, v \;.\; I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \Rightarrow G_1\text{-}PostCond(w,w')) \vee (G_2\text{-}Guard(w) \Rightarrow G_2\text{-}PostCond(w,w')))\}$$

This definition expresses that we start in a state w where both **EvG1** and **EvG2** can be fired; i.e. their guards holds. Hence, we can achieved the state w' if at least one event is executed. Notice that the "exclusive" characteristic between these two events is an additional constraint that will be considered further in the work. Based on this last definition, the two parts of the forward simulation condition can be stated as follows:

$$r^{-1}; (EvG1 \text{ OR } EvG2) \triangleq \{v \mapsto w' \mid \exists\, w \;.\; I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \Rightarrow G_1\text{-}PostCond(w,w')) \vee (G_2\text{-}Guard(w) \Rightarrow G_2\text{-}PostCond(w,w')))\} \\ EvG; r^{-1} \triangleq \{v \mapsto w' \mid \exists\, v'.I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w')\ \}$$

6.2.2 Proof obligations identification

The translation of the forward simulation condition, namely $r^{-1}; (EvG1 \text{ OR } EvG2) \subseteq EvG; r^{-1}$, requires to prove the following sequent:

$$I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \Rightarrow G_1\text{-}PostCond(w,w')) \vee (G_2\text{-}Guard(w) \Rightarrow G_2\text{-}PostCond(w,w'))) \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w')$$

The application of the inference rules: *OR-L* [4] then *IMP-L* then *MON*[5] allows us to simplify the above sequent by obtaining these two following sequents :

$$I(v) \wedge J(v,w) \wedge G_1\text{-}Guard(w) \wedge G_1\text{-}PostCond(w,w') \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w')$$

The definition of the OR goal refinement pattern given by [4] help us to detect the sufficient proof obligations in order to prove the last sequent; i.e. we ensure that proving these proof obligations is sufficient to prove the sequent:

$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \qquad \textbf{(PO1)} \\ G_1\text{-}PostCond \Rightarrow G\text{-}PostCond \qquad \textbf{(PO2)}$$

[4] It corresponds to the classical technique of a proof by cases. More precisely, in order to prove a goal under a disjunctive assumption $P \vee Q$, it is sufficient to prove independently the same goal under assumption P and also under assumption Q(see [3]).

[5] It says that in order to have a proof of goal G under the two sets of assumptions $H1$ and $H2$, it is sufficient to have a proof of G under $H1$ only (see [3]).

$$\boxed{\begin{array}{l} I(v) \wedge J(v,w) \wedge G_2\text{-}Guard(w) \wedge G_2\text{-}PostCond(w,w') \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}}$$

Similarly, the second sequent can be proved by proving the following proof obligations:

$G_2\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO3)**
$G_2\text{-}PostCond \Rightarrow G\text{-}PostCond$ **(PO4)**

The second part of the definition *(OR.def)* that express the exclusive characteristic of the OR refinement consists to prove these two proof obligations:

$$G_1\text{-}PostCond \Rightarrow \neg G_2\text{-}Guard \qquad \textbf{(PO5)}$$
$$G_2\text{-}PostCond \Rightarrow \neg G_1\text{-}Guard \qquad \textbf{(PO6)}$$

6.3 Synthesis

The OR goal refinement pattern is expressed in Event-B by considering that each concrete event (**EvG1** or **EvG2**) indeed refines its abstraction. However, we require an additional proof obligations ensuring that it is the realization of exactly one sub-goal which allow the realization of the parent goal:

- *The guard strengthening* **(PO1, PO3)** ensures that the concrete guard (G_1-*Guard* or G_2-*Guard*) is stronger than the abstract guard of **EvG**.
- *The correct refinement* **(PO2, PO4)** ensures that each concrete event (**EvG1** or **EvG2**) transforms the concrete variables in a way which does not contradict the abstract event **EvG**.
- *The "exclusive" characteristic* **(PO5, PO6)** ensures that only one event (either **EvG1** or **EvG2**) but not both can be executed.

Most of these proof obligations could be discharged by the current version of the Rodin automatic theorem prover [28]. In fact, this Event-B refinement semantics is quite close to the same one proposed by Rodin if we consider that each event refines the abstract event **EvG**.

7 Some other KAOS goal refinement patterns

The following patterns [4] introduce case conditions that guide the refinement of the parent goal.

7.1 The decomposition-by-case pattern.

This pattern [4] is applicable to behavioral Achieve goals where different cases can be identified for reaching the target condition **G-Post** . The cases must be disjoint and cover the entire state space. As shown in Figure 7, this refinement requires two domain properties (color shape), one stating the disjointness and coverage property of the case condition **Case1** and **Case2**, the other stating that the disjunction of the specific target conditions must imply the target condition of the parent goal. The completeness of the AND-refinement is derivable by use of those two domain properties.

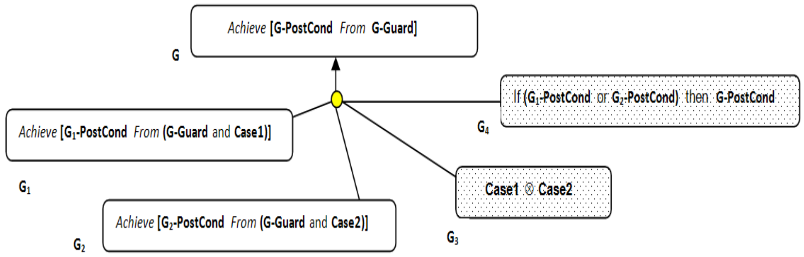

Achieve [G-PostCond From G-Guard]
G
If (G₁-PostCond or G₂-PostCond) then G-PostCond
G₄
Achieve [G₁-PostCond From (G-Guard and Case1)]
G₁
Case1 ⊗ Case2
G₃
Achieve [G₂-PostCond From (G-Guard and Case2)]
G₂

Figure 7: Decomposition-by-cases pattern

The study of this pattern reveals that A. Van Lamsweerde [4] gives it a very large definition. For this reason, we restrict the condition of using this pattern. Hence, we recommend requirements engineers to use this kind of pattern if we process just one element in the Achieve goal. In this case, we must verify that the cases are disjoint and cover the entire state space. This pattern can be seen as a special case of the OR refinement pattern since G_1-*Guard* and G_2-*Guard* are very large than *G-Guard* and can encapsulate the different cases **Case1** and **Case2**; i.e. these case are hidden in the guards G_1-*Guard* and G_2-*Guard*. Moreover, the proof obligations associated to the OR refinement allows us to express the two domain properties.

In the set-theoretic case, this pattern is not appropriate and have no sense. It is more judicious to use the AND refinement pattern (with disjoint variable) that can express what we want since we consider more large guards of sub-goals (G_1-*Guard* and G_2-*Guard* are very large than *G-Guard*). In fact, these guards can encapsulate the different cases **Case1** and **Case2**.

Consequently, we don't need to formalize the decomposition-by-case pattern in Event-B since our basic patterns allow us to express such pattern.

7.2 The guard-introduction pattern.

This pattern is a case-driven refinement pattern applicable to behavioral Achieve goals where a guard condition **Condition** must necessarily be set for reaching the target condition **G-Post**. As shown in Figure 8, the first sub-goal of the Achieve goal states that the target condition must be reached from a current condition **G-Guard** where in addition the guard condition **Condition** must hold. The second sub-goal states that the guard condition **Condition** must be reached as a target from that current condition **G-Guard**. The third sub-goal states that the current condition **G-Guard** must always remain true unless the target condition **G-Post** of the parent goal is reached.

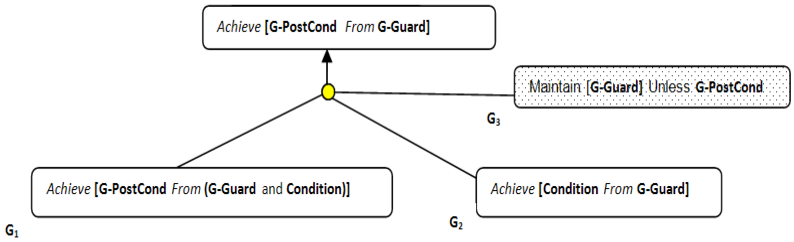

Achieve [G-PostCond From G-Guard]
Maintain [G-Guard] Unless G-PostCond
G3
Achieve [G-PostCond From (G-Guard and Condition)]
G1
Achieve [Condition From G-Guard]
G2

Figure 8: Guard-introduction pattern

If we switch the first two sub-goals, we remark that we obtain easily the milestone refinement pattern; i.e. if we consider of course that **G-Guard** must always remain true unless **G-Post** is reached (the third sub-goal). Hence, the guard-introduction pattern is a special case of the milestone refinement pattern. So, we don't need to formalize this pattern in Event-B.

8 Case Study

For the case study, we consider the specification of a localization software component which is a critical part of a land transportation system. Many positioning systems have been proposed over the last years. GPS, one of the most widely used positioning system, is perhaps the best-known. This system belongs to the GNSS (Global Navigation Satellite Systems) family which also regroups GALILEO or GLONASS. Positioning systems are often dedicated to a particular environment; the GNSS technology, for example, generally does not work indoors. To resolve these problems, numerous alternatives relying on very different technologies have arisen. Those last years, Wireless LAN such as IEEE 802.11 networks have been considered by numerous location systems. These systems all use the radio signal strength to determine the physical location. Localization systems can therefore be designed using various technologies like wireless personal networks such as Wifi or Bluetooth [25, 26], sensors [27], GNSS repeaters or visual landmarks.

The main difficulties when we develop a localization component is to find the correct algorithm that merges positioning data and to take into account all the properties we have to deal with. At this stage, we think that a semi-formal model will be very useful to have guidelines on how to do. However, elaborating this semi-formal model is not necessarily an easy task. Often, requirements engineers need to search through preliminary documents in order to extract goals (key properties) using a number of heuristics (asking HOW and WHY questions...) detailed in [4]. Figure 9 show the obtained KAOS goal model of a localization component thanks to heuristics. For example, a HOW question about the goal G would then lead to the goals G_1, G_2 and G_3. This obtained KAOS goal model contains: (i) high-level goals; (ii) refinement links denoted by a bubble linking the parent goal with an arrow, and the child goals with regular lines; (ii) requirements and their software responsibility agents; (iv) expectations and their environment responsibility agents (GPS, WIFI, sensor and accelerometer).

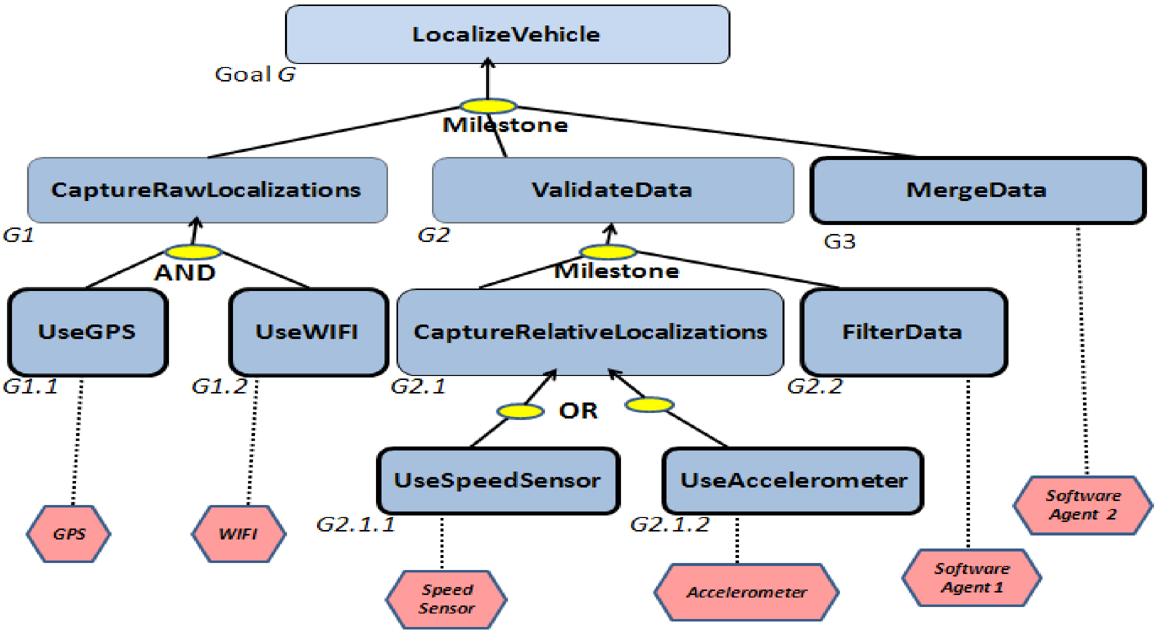


Figure 9: KAOS goal model of a localization component

Let us start by the high level goal G which is defined as follows:

Goal G: Achieve [LocalizeVehicle]
InformalDef: The Cycab/vehicle must be localized.

We associate an Event-B model *Localization*, in Figure 10, to this most abstract level of the hierarchy of the KAOS goal graph. In this Event-B model, we will have an event called **LocalizeVehicle** that will translate the goal G; i.e. it describes the "property" of the goal G, in terms of generalized substitutions. Localizing a vehicle consists in obtaining an *estimated_loc* which is a couple of latitude and longitude. At this level of abstraction, it is not necessary to precise the way this information is obtained. Thus we use the non-deterministic generalized substitution through the symbol $:\in$. It specifies an unbounded choice and *estimated_loc* can take any value in the sets $(LATITUDE \setminus \{null\})$ and $(LONGITUDE \setminus \{null\})$. The *null* value serves just to initialize the system through the **initialization** event[6]. Notice that at this abstraction level, the event **LocalizeVehicle** can always occurs. Hence, its guard is set to *true*. Sets in uppercase are abstract sets used to type the variables. They are described in the Event-B context *TypeSets* (see in Figure 11).

[6] The initialization part, variables, invariants and contexts are manually completed by the designer.

MACHINE Localization
SEES TypeSets
VARIABLES

  estimated_loc

INVARIANTS

  inv1 : $estimated\_loc \in LATITUDE \times LONGITUDE$

EVENTS
Initialisation

  begin
    act1 : $estimated\_loc := null \mapsto null$
  end

Event *LocalizeVehicle* $\widehat{=}$

  begin
    act1 : $estimated\_loc :\in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})$
  end

END

Figure 10: The abstract model

CONTEXT TypeSets
SETS

  SUBCOMPONENTS

  SUBSENSORS

CONSTANTS

  gps, wifi, LATITUDE, LONGITUDE

  null, speed, accel

AXIOMS

  axm1 : $partition(SUBCOMPONENTS, \{gps\}, \{wifi\})$
  axm2 : $LATITUDE = \mathbb{N} \cup \{null\}$
  axm3 : $LONGITUDE = \mathbb{N} \cup \{null\}$
  axm4 : $partition(SUBSENSORS, \{speed\}, \{accel\})$

END

Figure 11: The Event-B context *TypeSets*

8.1 First refinement

The goal G is refined into three sub-goals according to the milestone goal refinement pattern:

Goal G_1: Achieve [CaptureRawLocalizations]

InformalDef: Firstly, several sets of raw localization data are captured by using different technologies.

Goal G_2: Achieve [ValidateData]

InformalDef: Then, all the sets of raw localization data will be validated and controlled.

Goal G_3: Achieve [MergeData]

InformalDef: Finally, all the validated data will be merged in order to obtain the final localization.

Similarly, we associate an Event-B refinement model *Localization1*, in Figure 12, to this first level of the hierarchy of the KAOS goal graph. The sub-goals G_1, G_2 and G_3 are represented by three Event-B events **CaptureRawLocalizations**, **ValidateData** and **MergeData**, respectively. The first one returns a set of couples (latitude, longitude), one for each component used for localizing a vehicle. The second one validates the returned set of couples by choosing the acceptable values. The final one returns the final localization calculated from the returned values of the event **MergeData**.

In Event-B, often the information about such event ordering has to be embedded into guards and event actions with the downside of extra model variables. As we said previously, we have chosen to explicitly reproduce KAOS goal ordering in an Event-B model by proposing a syntactic extension of the Event-B refinement proof rule in order to provide a way to refine an abstract event by a sequence of new events. Hence, the abstract event **LocalizeVehicle** is refined as follows:

(**CaptureRawLocalizations**; **ValidateData**; **MergeData**) Refines **LocalizeVehicle**

In addition to *the feasibility proof obligation*[7], this kind of refinement requires to discharge these different proof obligations:

- *Two ordering constraints* express the "milestone" characteristic between the Event-B events. These two proof obligation are discharged: (i) the action of **CaptureRawLocalizations** implies the guard of **ValidateData** (ii) the action of **ValidateData** implies the guard of **MergeData**.
- *One "guard strengthening"* is also discharged since the first event (**CaptureRawLocalizations**) in the sequence has a guard (*true*) that implies the abstract guard (*true*).
- *One "correct refinement"* is also proved since the last event (**MergeData**) in the sequence has a postcondition that implies the abstract postcondition under the gluing invariant $inv4$. Hence, the sequence of concrete events transforms the concrete variables in a way which does not contradict the abstract event.

[7] It ensures that each event must also be feasible, in a sense that an appropriate new state v' must exist for some given current state v.

```
MACHINE  Localization1
REFINES  Localization
SEES  TypeSets
VARIABLES
    estimated_loc, subcomponents_loc, validated_loc, merged_loc
INVARIANTS
    inv1 : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow (LATITUDE \times LONGITUDE)$
    inv2 : $validated\_loc \in SUBCOMPONENTS \nrightarrow (LATITUDE \times LONGITUDE)$
    inv3 : $merged\_loc \in LATITUDE \times LONGITUDE$
    inv4 : $estimated\_loc = merged\_location$
EVENTS
Initialisation
    begin
        act1 : estimated_loc := null ↦ null
        act4 : $subcomponents\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$
        act3 : $validated\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$
        act5 : $merged\_loc := null \mapsto null$
    end
Event  CaptureRawLocalizations ≙
    begin
        act1 : $subcomponents\_loc :\in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
    end
Event  ValidateData ≙
    when
        grd1 : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
    then
        act1 : $validated\_loc :\in \mathcal{P}_1(subcomponents\_loc)$
    end
Event  MergeData ≙
    when
        grd1 : $validated\_loc \in \mathcal{P}_1(subcomponents\_loc)$
        grd2 : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
    then
        act1 : $merged\_loc :\in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})$
    end
END
```

Figure 12: First Event-B refinement model

8.2 Second refinement

Now, we consider the second level of the hierarchy of the KAOS goal graph. In the same way, a refinement Event-B model *Localization2*, in Figure 13, is created and must refine the previous model *Localization1*. This second refinement Event-B model will encapsulate two KAOS refinement patterns:

8.2.1 Second refinement: Applying the AND goal refinement pattern

The goal $G1$ is AND-refined into two sub-goals. This refinement specifies the kind of technology used to obtain localization data.

Goal $G_{1.1}$: Achieve [UseGPS]
InformalDef: A GPS system is used.

Goal $G_{1.2}$: Achieve [UseWIFI]
InformalDef: A wireless technique is used.

The sub-goals $G_{1.1}$ and $G_{1.2}$ are represented by two Event-B events **UseGPS** and **UseWIFI** by using the same transcription rules as for the event **LocalizeVehicle** in the abstract model. Since the goal G_1 is refined into two sub-goals $G_{1.1}$ and $G_{1.2}$ according to the AND goal refinement pattern, the execution of the corresponding new events must not necessary follows a specific order. As we said previously, our idea (inspired from Process Algebra [21]) is that these events (**UseGPS** and **UseWIFI** are executed in an arbitrary order: either **UseGPS;UseWIFI** or **UseWIFI;UseGPS** as follows (of course, we must ensure that this AND refinement manipulate only *disjoint set of variables*):

(**UseGPS** ||| **UseWIFI**) Refines **CaptureRawLocalizations**

In addition to *the feasibility proof obligation*, the following proof obligations must be discharged in order to prove such refinement:

- *Two "guard strengthening"* are discharged since the concrete guard of (**UseGPS** ||| **UseWIFI**) implies the abstract guard (*true*) of **CaptureRawLocalizations**. In fact, the concrete guard is always *true* (if we execute **UseGPS** at first or if we execute **UseWIFI** at first).
- *One "correct refinement"* is also proved since the conjunction of the two concrete postconditions implies the abstract postcondition under the gluing invariant $inv3$. Hence, this ensures that *subcomponents_loc* is a total function (see the postcondition of the abstract event **CaptureRawLocalizations**).

8.2.2 Second refinement: Applying the milestone goal refinement pattern

On the other hand, the goal G_2 is refined into two sub-goals according to the milestone-driven tactics:

Goal $_{2.1}$: Achieve [CaptureRelativeLocalizations]
InformalDef: At first, several sets of relative localization data are captured by using

MACHINE Localization2

REFINES Localization1

SEES TypeSets

VARIABLES

$\texttt{estimated\_loc}, \texttt{subcomponents\_loc}, \texttt{validated\_loc}, \texttt{merged\_loc}$

$\texttt{gps\_loc}, \texttt{wifi\_loc}, \texttt{sensors\_loc}, \texttt{kept\_loc}$

INVARIANTS

inv1 : $gps\_loc \in \{gps\} \longrightarrow (LATITUDE \times LONGITUDE)$

inv2 : $wifi\_loc \in \{wifi\} \longrightarrow (LATITUDE \times LONGITUDE)$

inv3 : $subcomponents\_loc = gps\_loc \cup wifi\_loc$

inv4 : $sensors\_loc \in SUBSENSORS \mathrel{+\!\!\!\to} (LATITUDE \times LONGITUDE)$

inv5 : $kept\_loc \in SUBCOMPONENTS \mathrel{+\!\!\!\to} (LATITUDE \times LONGITUDE)$

inv6 : $validated\_loc = kept\_loc$

EVENTS

Initialisation

begin

act1 : $\texttt{estimated\_loc} := \texttt{null} \mapsto \texttt{null}$

act4 : $\texttt{subcomponents\_loc} :\in \texttt{SUBCOMPONENTS} \longrightarrow (\{\texttt{null}\} \times \{\texttt{null}\})$

act3 : $\texttt{validated\_loc} :\in \texttt{SUBCOMPONENTS} \longrightarrow (\{\texttt{null}\} \times \{\texttt{null}\})$

act5 : $\texttt{merged\_loc} := \texttt{null} \mapsto \texttt{null}$

act7 : $gps\_loc :\in \{gps\} \longrightarrow (\{null\} \times \{null\})$

act6 : $wifi\_loc :\in \{wifi\} \longrightarrow (\{null\} \times \{null\})$

act8 : $sensors\_loc :\in SUBSENSORS \longrightarrow (\{null\} \times \{null\})$

act9 : $kept\_loc :\in SUBCOMPONENTS \longrightarrow (\{null\} \times \{null\})$

end

Event *UseGPS* $\widehat{=}$

begin

act1 : $gps\_loc :\in \{gps\} \longrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

end

Event *UseWIFI* $\widehat{=}$

begin

act1 : $wifi\_loc :\in \{wifi\} \longrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

end

Event *CaptureRelativeLocalizations* $\widehat{=}$

when

grd1 : $subcomponents\_loc \in SUBCOMPONENTS \longrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

then

act1 : $sensors\_loc \;:\; |(sensors\_loc' \in SUBSENSORS \mathrel{+\!\!\!\to} ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))) \wedge sensors\_loc' \neq \varnothing$

end

Event *FilterData* $\widehat{=}$

when

grd1 : $sensors\_loc \in SUBSENSORS \mathrel{+\!\!\!\to} ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})) \wedge sensors\_loc \neq \varnothing$

then

act1 : $kept\_loc :\in \mathcal{P}_1(subcomponents\_loc)$

end

END

Figure 13: Second Event-B refinement model

different technologies.

Goal $_{2.2}$: Achieve [FilterData]
InformalDef: Then, all the sets of raw localization data will be filtered.

All these subgoals are translated into new events using the same rules as for **LocalizeVehicle** in the abstract Event-B model. As for the first refinement, the abstract event **ValidateData** is refined by the *sequence* of all the new events (**CaptureRelativeLocalizations**, **FilterData**) as follows:

(**CaptureRelativeLocalizations** ; **FilterData**) Refines **ValidateData**

We have also discharged the different proof obligations related to the milestone refinement such as *the ordering constraint*, *the "guard strengthening"* and *the "correct refinement"*.

8.3 Third refinement

The goal $G_{2.1}$ is OR-refined in two subgoals:
Goal $G_{2.1.1}$: Achieve [UseSpeedSensor]
InformalDef: The Cycab may use a speed sensor system.

Goal $G_{2.1.2}$: Achieve [UseAccelerometer]
InformalDef: Or, it may use the accelerometer system.

Similarly, a refinement Event-B model *Localization3*, in Figure 14, is associated to this third level of the hierarchy of the KAOS goal graph. All these subgoals are transformed into new Event-B events (**UseSpeedSensor**, **UseAccelerometer**) by using the same transcription rules as for the event **LocalizeVehicle** in the abstract Event-B model. Since the satisfaction of exactly one KAOS sub-goal implies the satisfaction of the parent goal, we propose to refine the abstract event **CaptureRelativeLocalizations** as follows:

(**UseSpeedSensor** XOR **UseAccelerometer**) Refines **CaptureRelativeLocalizations**

As we said previously, this Event-B refinement semantics is quite close to the same one proposed by Rodin [28] if we consider that each event refines the abstract event. Hence, the proof obligations ("guard strengthening" and "correct refinement") could be discharged by the current version of the Rodin automatic theorem prover [28]. However, we require to express two additional proof obligations ensuring that only one event (either **UseSpeedSensor** or **UseAccelerometer**) but not both can be executed. These two proof obligations are discharged since (i) the postcondition of **UseSpeedSensor** forbids the guard of **UseAccelerometer** to be triggered; (ii) the postcondition of **UseAccelerometer** forbids the guard of **UseSpeedSensor** to be triggered.

This is the last step of refinement since all the goals are either *requirements* or *expectations*. This allows to obtain the abstract Event-B specification from the KAOS goal hierarchy. The Event-B model is then further refined towards an implementation. It concerns only the Event-B events corresponding to requirements assigned to a software argent. Otherwise, an expectation (assigned to an external agent) is a property required on the environment and it corresponding Event-B event will not be implemented in the software-to-be. More precisely, in our case study, the Event-B events **UseGPS**, **UseWIFI**, **UseSpeedSensor**

MACHINE Localization3
REFINES Localization2
SEES TypeSets
VARIABLES

    sensors_loc, speed_loc, accel_loc

INVARIANTS

    inv1 : $speed\_loc \in \{speed\} \rightarrow (LATITUDE \times LONGITUDE)$
    inv2 : $accel\_loc \in \{accel\} \rightarrow (LATITUDE \times LONGITUDE)$
    inv3 : $(sensors\_loc = speed\_loc) \lor (sensors\_loc = accel\_loc) \lor (sensors\_loc = speedloc \cup accel\_loc)$

EVENTS
Initialisation

    begin
        act10 : $speed\_loc :\in \{speed\} \rightarrow (\{null\} \times \{null\})$
        act11 : $accel\_loc :\in \{accel\} \rightarrow (\{null\} \times \{null\})$
    end

Event *UseSpeedSensor* $\widehat{=}$
refines *CaptureRelativeLocalizations*

    when
        grd1 : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
        grd2 : $accel\_loc \in \{accel\} \rightarrow (\{null\} \times \{null\})$
    then
        act1 : $speed\_loc, sensors\_loc : |(speed\_loc' \in \{speed\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))) \land sensors\_loc' = speed\_loc'$
    end

Event *UseAccelerometer* $\widehat{=}$
refines *CaptureRelativeLocalizations*

    when
        grd1 : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
        grd2 : $speed\_loc \in \{speed\} \rightarrow (\{null\} \times \{null\})$
    then
        act1 : $accel\_loc, sensors\_loc : |(accel\_loc' \in \{accel\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))) \land sensors\_loc' = accel\_loc'$
    end

END

Figure 14: Third Event-B refinement model

```
  lat, long ← OPMerge =
    VAR lat_gps, long_gps, lat_wifi, long_wifi, ponderation_gps, ponderation_wifi
    IN
      lat_gps := get_lat(gps_loc)||
      long_gps := get_long(gps_loc) ||
      ponderation_gps := get_pond(gps_loc) ||
      lat_wifi := get_lat(wifi_loc)||
      long_wifi := get_long(wifi_loc) ||
      ponderation_wifi := get_pond(wifi_loc) ;

        lat := ((lat_gps ∗ ponderation_gps) + (lat_wifi ∗ ponderation_wifi))          /
(ponderation_gps + ponderation_wifi) ||
      long := ((long_gps ∗ ponderation_gps) + (long_wifi ∗ ponderation_wifi))         /
(ponderation_gps + ponderation_wifi)
  END
```

Figure 15: The B operation **OPMerge**

and **UseAccelerometer** are not refined since they correspond to goals of type *expectation*. They are implemented by hardware components (GPS, WIFI components...) in a vehicle. Only the Event-B events **FilterData** and **MergeData** are further refined. For example, the refinement of the Event-B event **MergeData** leads to a software that implements the algorithm chosen to realize the fusion (thanks to the B operation **OPMerge**) as shown in Figure 15. At first, this operation recovers all the raw localization data from both GPS and WIFI thanks to the call of operations *get_lat* and *get_long*. Then, the call of the operation *get_pond* serves to verify if the returned values of GPS and WIFI are validated or no. If these values are validated, so *get_pond* returns a weighting value set to 1 (*O* otherwise). Finally, the operation calculates the final latitude and longitude based on the different weighting values.

An interesting result is that the link between the B operation **OPMerge** and the abstract Event-B event **MergeData** (see Figure 12) is ensured. While the abstract event **MergeData** describes the properties that the final program must fulfill, the B operation **OPMerge** describes the algorithm contained in the program. Hence, **MergeData** describes the way by which we can eventually judge that the final program **OPMerge** is correct: (i) the call of the operations *get_lat* and *get_long* ensures the second guard of **MergeData**; (ii) the call of the operation *get_pond* ensures the first guard of **MergeData**; (iii) the final result of the operation **OPMerge** *(lat, long)* satisfies the post-condition of **MergeData**.

9 Discussion

One may wonder whether the formalization of KAOS target conditions as Event-B postconditions is adequate, since the execution of Event-B events is not mandatory. In accordance with the Event-B semantics, all events whose guard is true can be performed and there is necessarily one that will be performed. The choice between all these permitted events is made non-deterministically.

Another important point is that up to now we have not studied the combination and the interaction between the different KAOS refinement pattern. Consequently, we are not certain to maintain all the proof obligations. For instance, the study of the global behavior of the KAOS goal graph, thanks to trace semantics, reveals the following main problem caused by the strong proof obligation **PO1** of the milestone refinement pattern (see Section 4). Let us explain this problem by considering the following goal model (Figure 16):

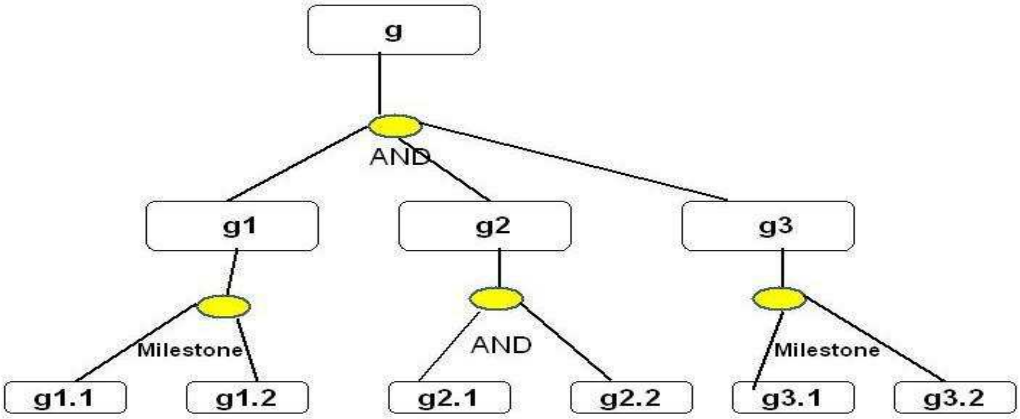


Figure 16: KAOS goal model

According to our Event-B refinement semantics, the events of the first refinement are interleaved as follows:

$$ev1|||ev2|||ev3$$

Now, one of possible valid execution trace in the second refinement is :

$$ev1.1 \ ; ev2.1 \ ; ev2.2 \ ; ev1.2 \ ; ev3.1 \ ; ev3.2$$

The main problem is that the execution of $ev2.1$ or $ev2.2$ can imply that the guard of $ev1.2$ become false. So, the proof obligation "ordering constraint" of the milestone refinement pattern ($ev1.1$-Post $\Rightarrow$ $ev1.2$-Guard) can never be discharged in this case. Indeed, it is more logical now to relax this proof obligation by updating it as follows:

$$\Box(ev1.1\text{-Post} \Rightarrow \Diamond\ ev1.2\text{-Guard})$$

This property means that it is always the case that once $ev1.1 - Post$ holds then $ev1.2 - Guard$ holds eventually. Notice that $ev1.1 - Post$ is only required to hold at the initialization of this process. So, $ev1.1 - Post$ is not required to hold any more before achieved $g1.2 - Guard$. This semantic corresponds to the classical temporal operator [22] *leadsto* ($\leadsto$) as follows:

$$ev1.1 - Post \leadsto ev1.2 - Guard$$

Proof techniques as in the Event-B method cannot be used to verify such ordering constraints. It seems that model checking may be a good alternative to prove such properties that contain a temporal operator. So, our aims is to mix "local" proof obligations (generated by the prover) with the proof generated by the model checker. We think that a tool like ProB [15] may be a very useful tool since it helps to discover the faults, where the consistency is not completely achieved by the B prover.

The interest of the proposed approach is that we can prove some properties of consistency on the goal model by discharging the proof obligations generated by the Event-B refinement process. Indeed, if we can discharge for instance the proof obligations of refinement of an event **EvG** by either **EvG1** or **EvG2** it means that the OR decomposition of goal G is correct. Otherwise, it means either that one or more expressions of the events (**EvG**, **EvG1**,

EvG2) are not correct or that sub-goals are missing or that the goal refinement pattern is false. However, we can never ensure that the expression of the Event-B events corresponds exactly to the expression of the related goal since this later is informal. This means that proving the Event-B formal counterpart of the KAOS goal model is not sufficient in order to validate the conformance of the specification to original requirements. For that, we can use an animation technique to validate the derived formal specification and consequently its semi-formal counterpart goal model against original customers' requirements. This animation step not only indicates deviations from original requirements right on the spot but also helps fixing some specification errors. The reader can refer to [17] for more details. For that, ProB [15] may be also a very useful validation tool since its automated animation facilities allow users to animate their specifications.

10 Related work

Our proposed approach aims at expressing KAOS models with Event-B language by staying at the same abstraction level. In the sequel, we outline a number of approaches that have tried to bridge the gap between KAOS requirements model and formal methods. With the best of our knowledge, they are the only work that deal with such a problematic.

The work of [13] presents a goal-oriented approach to elaborate a pertinent model and turn it into a high quality abstract B machines. The scheme used by the authors for transforming the KAOS requirements model to B is as follows: As agents are the active entities able to perform operations, a B machine is associated with each KAOS agent. The agent attributes and the operations arguments are represented by the sets, variables and constraints. All *maintain* goals under the agent responsibility are translated as invariants of the corresponding B machine. All the KAOS operations that an agent has to perform are represented by B operations.

The authors of [9] provides means for transforming the security requirements model built with KAOS to an equivalent one in B. This abstract B model is then refined using non-trivial B refinements that generate design specifications conforming to the initial set of security requirements. The authors consider each operational goal and each KAOS operation as a B operation. Also, they consider that each KAOS object, related to the operational goals, is B machine. So, The relationship among objects is captured using the B machine imports, includes, uses, and sees clauses that allow one B machine to relate to or compose other B machines. KAOS domain properties are transformed to B invariants or pre-conditions related to the corresponding B operations. The authors introduce the concept of goal achievement which is reflected through the return values of the B operations that model KAOS goals. Hence, each B operation corresponding to an operational goal returns a flag indicating whether the goal implemented in this operation has been achieved or not.

We can also point out a work [12] proposing an automatically generator that transforms an extend KAOS model into VDM++ specifications. The generator connects operations in KAOS to those in VDM++, and entities in KAOS to objects or types in VDM++. The generated specification contains implicit operations consisting of pre- and post-conditions, inputs, and outputs of operations. However, this generated specifications require software developers to add the body of operations in order to create explicit specifications.

The GOPCSD (Goal-oriented Process Control System Design) tool [8] is an adaptation of the KAOS method that serves to analyze the KAOS requirements and generate B formal specifications. The tool is used to construct the application requirements in the form of goal-models by interacting with the user and importing library templates. Then, the requirements are checked to enable the system engineer to debug and correct them. Finally, the requirements will be translated to B specifications. The generated specifications can be refined and translated to executable code by a software engineer.

Recently, [16] presents a constructive verification-based approach that consists in linking requirements, expressed as linear temporal logic formulae, to a system specification expressed as an Event-B machine extended with the notion of obligations [14]. The source requirements are included as verification assertions that can be model-checked by tools like ProB [15], showing that the proposed specification indeed meets the system requirements.

Nevertheless, the reconciliation presented by all of these works remains partial because they don't consider all the parts of the KAOS goal model but only the requirements (operational goals). Consequently, the formal model does not include any information about the non-operational goals and, more important, the type of goal refinement. In this paper, we have explored how to cope with this problem using an approach that expresses the whole KAOS goal model with a formal method like Event-B by staying at the same abstraction level. Our approach can be considered as complementary to existing ones. Furthermore, what we present can be very useful in practice to (i) systematically verify that all KAOS requirements are represented in the Event-B model; (ii) systematically verify that each element in the Event-B model has a purpose in KAOS. Moreover, employing formal methods in requirements engineering level allows us to detect anomalies when we use the goal refinement pattern in a chaotic manner. Hence, formal methods offers a recommendation support to requirement engineers in order to choose the appropriate goal refinement patterns.

11 Conclusion and further work

In this report, we have presented a constructive approach driven by goals showing that it is possible to express KAOS goal models with Event-B. Even if formal methods are harder to use and less widely applicable, we have shown that extending KAOS with more formality provide much higher precision and richer forms of analysis. The main contribution of our approach is that it establishes the first brick toward the construction of the bridge between the non-formal and the formal worlds as narrow and concise as possible. This brick balances the trade-off between complexity of rigid formality (Event-B method) and expressiveness of semi-formal approaches (KAOS). However, a number of future research steps are ongoing. Regarding the different KAOS goal model concepts, we need now to consider the translation of the concepts of domain properties and non functional goals. We plan also to apply the approach on a number of case studies. Moreover, it would be interesting to establish the correspondence between the obtained Event-B representation of KAOS goal models and the later phases of development. At tool level, we plan to develop a connector between a KAOS toolkit and the RODIN open platform.

References

[1] J.R. Abrial. The B-Book: Assigning programs to meanings. Cambridge University Press, 1996.

[2] J.R. Abrial. Extending B without changing it (for developing distributed systems). The First Conference on the B-Method,pages 169–190, IRIN, Nantes, France, 1996.

[3] J.R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.

[4] A. van Lamsweerde. Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[5] R. Darimont and A. van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In *SIGSOFT '96*, pages 179–190, San Francisco, California, USA, October 1996. ACM SIGSOFT.

[6] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. METEOR : A successful application of B in a large project. In *FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems*, Volume I pages 369–387, 1999. Springer.

[7] F. Badeau and A. Amelot. Using B as a high level programming language in an industrial project: Roissy val. In *Proceedings of the 4th International Conference of B and Z Users (ZB'05)*, pages 334–354, Guildford, UK, 2005. Springer.

[8] I. El-Madah and T. Maibaum. Goal-Oriented Requirements Analysis for Process Control Systems Design. In *MEMOCODE 2003*, pages 45–46, Mont Saint-Michel, France, June 2003. IEEE Computer Society.

[9] R. Hassan and S. Bohner and S. El-Kassas and M. Eltoweissy. Goal-Oriented, B-Based Formal Derivation of Security Design Specifications from Security Requirements. In *ARES 2008*, pages 1443–1450, Barcelona, Spain, March 2008. IEEE Computer Society.

[10] A. Matoussi and F. Gervais and R. Laleau A First Attempt to Express KAOS Refinement Patterns with Event B. In *ABZ 2008*, pages 338, London, UK, September 2008. Springer.

[11] A. Matoussi and F. Gervais and R. Laleau De KAOS vers Event-B : Approche dirigée par les buts. In *Actes de la conférence AFADL'2009*, pages 71–86, Toulouse, France, Janvier 2009.

[12] H. Nakagawa and K. Taguchi and S. Honiden. Formal Specification Generator for KAOS: Model Transformation Approach to Generate Formal Specifications from KAOS Requirements Models. In *ASE 2007*, pages 531–532, Atlanta, Georgia, USA, November 2007. ACM.

[13] C. Ponsard and E. Dieul. From Requirements Models to Formal Specifications in B. In *REMO2V'2006*, Luxembourg, June 2006.

[14] J. Bicarregui and A. Arenas and B. Aziz and P. Massonet and C. Ponsard. Towards Modelling Obligations in Event-B. In *ABZ 2008*, pages 181–194, London, UK, September 2008. Springer.

[15] M. Leuschel and M.J. Butler. ProB: A Model Checker for B. In *K. Araki, S. Gnesi, D. Mandrioli (eds), FME 2003: Formal Methods, LNCS 2805*, pages 855–874, 2003. Springer.

[16] B. Aziz and A. Arenas and J. Bicarregui and C. Ponsard and P. Massonet. From Goal-Oriented Requirements to Event-B Specifications. In *In: First Nasa Formal Method Symposium (NFM 2009)* , Moffett Field, California , USA, April 2009.

[17] A. Mashkoor and A. Matoussi Towards Validation of Requirements Models. In *2nd International Conference on Abstract State Machines (ASM), Alloy, B and Z (ABZ'10)*, Orford, Canada, 2010. To appear.

[18] A. Mammar and R. Laleau A formal approach based on UML and B for the specification and development of database applications. In *Automated Software Engineering 13(4)*, pages 497-528, 2006.

[19] C. Snook and M. Butler. UML-B: formal modelling and design aided by UML. In *ACM Transactions on Software Engineering and Methodology, 15(1)*, pp. 92-122, 2006.

[20] E. Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *RE'97*, pages 226-235, 1997. IEEE Computer Society.

[21] D. Sangiorgi. Locality and interleaving semantics in calculi for mobile processes. In *Theor. Comput. Sci.*, Volume 155, pages 39–83, 1996. Elsevier Science Publishers Ltd.

[22] Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. Sci. Comput. Program., Volume 4, number 3, pages 257–289, 1984. Elsevier North-Holland, Inc.

[23] N. Lynch and F. Vaandrager. Forward and Backward Simulations - Part I: Untimed Systems. *Information and Computation Journal*, Volume 121, pages 214–233, 1994.

[24] W.N. Robinson and S. Pawlowski. Surfacing Root Requirements Interactions from Inquiry Cycle Requirements Documents. In *The Third IEEE International Conference on Requirements Engineering (ICRE'98)*, pages 82–89, Colorado Springs, CO, USA, 1998. IEEE Computer Society Press.

[25] J. Hallberg and M. Nilsson and K. Synnes. Positioning with bluetooth. In *10th Int. Conference on Telecommunications (ICT'2003)*, pages 954—958, 2003.

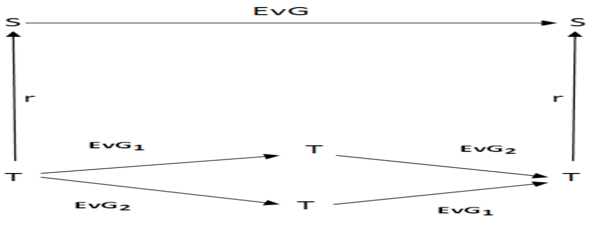
[26] J.A. Royo and E. Mena and L.C. Gallego. Locating Users to Develop Location-Based Services in Wireless Local Area Networks. In *UCAmI2005*, pages 471-478, Granada, Spain, 2005.

[27] R.J. Orr and G.D. Abowd. The smart floor: A mechanism for natural user identification and tracking. In *Conference on Human Factors in Computing Systems (CHI2000)*, pages 1–6, 2000. ACM Press

[28] RODIN - Rigorous Open Development Environment for Complex Systems. *http://rodin.cs.ncl.ac.uk/*

Appendix A. Another trace semantics to discover proof obligations related to the AND refinement

The proposed semantics consists to describe all the possible behaviors in terms of trace as illustrated in the following diagram. Each behavior is described in the form of sequence of events like the milestone refinement pattern.



Consequently, we have the following "forward simulation" proof to perform in order to ensure that any trace of a refined model must also be a trace of the abstraction:

$$(EvG1; EvG2) \text{ or } (EvG2; EvG1)$$

For that, we extend the classical set theoretic representation (see Figure **??**) as follows:

$$\boxed{\begin{array}{l} EvG1|||EvG2 \triangleq \{w \mapsto w' \,|\, \exists\, v \;.\; I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \Rightarrow \exists\, t \;.\; G_1\text{-}PostCond(w,t) \wedge \mathrm{G}_2\text{-}Guard(t) \wedge G_2\text{-}PostCond(t,w')) \vee \\ (G_2\text{-}Guard(w) \Rightarrow \exists\, t \;.\; G_2\text{-}PostCond(w,t) \wedge \mathrm{G}_1\text{-}Guard(t) \wedge G_1\text{-}PostCond(t,w')))\} \end{array}}$$

Consequently, the two parts of the forward simulation condition can be stated as follows:

$$\begin{array}{l} r^{-1}; (EvG1|||EvG2) \triangleq \{v \mapsto w' \,|\, \exists\, w \;.\; I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \Rightarrow \exists\, t \;.\; G_1\text{-}PostCond(w,t) \wedge \mathrm{G}_2\text{-}Guard(t) \wedge G_2\text{-}PostCond(t,w')) \vee \\ (G_2\text{-}Guard(w) \Rightarrow \exists\, t \;.\; G_2\text{-}PostCond(w,t) \wedge \mathrm{G}_1\text{-}Guard(t) \wedge G_1\text{-}PostCond(t,w')))\} \\ EvG; r^{-1} \triangleq \{v \mapsto w' \,|\, \exists\, v'.I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w')\ \} \end{array}$$

Consequently, the translation of the forward simulation condition, namely $r^{-1}; (EvG1|||EvG2) \subseteq EvG; r^{-1}$, requires to prove the following sequent:

$$\boxed{\begin{array}{l} I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ ((G_1\text{-}Guard(w) \;\Rightarrow\; \exists\, t \;.\; G_1\text{-}PostCond(w,t) \wedge \mathrm{G}_2\text{-}Guard(t) \;\wedge\; G_2\text{-}PostCond(t,w')) \vee \\ (G_2\text{-}Guard(w) \;\Rightarrow\; \exists\, t \;.\; G_2\text{-}PostCond(w,t) \wedge \mathrm{G}_1\text{-}Guard(t) \;\wedge\; G_1\text{-}PostCond(t,w')))\} \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}} \qquad \text{(A.I)}$$

The application of the inference rule *OR-L* allows us to simplify the above sequent by obtaining the two following sequents :

$$\boxed{\begin{array}{l} I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ (G_1\text{-}Guard(w) \;\;\Rightarrow\;\; \exists\, t \;\;.\;\; G_1\text{-}PostCond(w,t) \;\wedge\; G_2\text{-}Guard(t) \;\wedge\; G_2\text{-} \\ PostCond(t,w')) \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}} \qquad \text{(A.I.1)}$$

$$\boxed{\begin{array}{l} I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge \\ (G_2\text{-}Guard(w) \;\;\Rightarrow\;\; \exists\, t \;\;.\;\; G_2\text{-}PostCond(w,t) \;\wedge\; G_1\text{-}Guard(t) \;\wedge\; G_1\text{-} \\ PostCond(t,w')) \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}} \qquad \text{(A.I.2)}$$

For each sequent ((A.I.1) ,(A.I.2)), we apply (in order) the inference rules: *IMP-L* then *MON* then *XST-L*[8] in order to simplify the hypothesis part of the sequent and also removing useless hypotheses as follows:

$$\boxed{\begin{array}{l} I(v) \wedge J(v,w) \wedge \\ (G_1\text{-}Guard(w) \wedge G_1\text{-}PostCond(w,t) \wedge G_2\text{-}PostCond(t,w')) \\ \vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}} \qquad \text{(A.I.1')}$$

The definition of the AND goal refinement pattern given by [4] help us to detect the sufficient proof obligations in order to prove the last sequent; i.e. we ensure that proving these proof obligations is sufficient to prove the sequent:

$G_1\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO1)**
$(G_1\text{-}PostCond \wedge G_2\text{-}PostCond) \Rightarrow G\text{-}PostCond$ **(PO2)**

$$\boxed{\begin{array}{l} I(v) \wedge J(v,w) \wedge \\ (G_2\text{-}Guard(w) \wedge G_2\text{-}PostCond(w,t) \wedge G_1\text{-}PostCond(t,w')) \;\vdash \\ I(v) \wedge G\text{-}Guard(v) \wedge \exists\, v'.G\text{-}PostCond(v,v') \wedge I(v') \wedge J(v',w') \end{array}} \qquad \text{(A.I.2')}$$

Similarly, the second sequent comes down to prove the following sufficient proof obligations:

$G_2\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO3)**
$(G_2\text{-}PostCond \wedge G_1\text{-}PostCond) \Rightarrow G\text{-}PostCond$ **(PO2)**

[8] allows us to replace an existential assumption by one without the existential quantifier. This can only be done however if the quantified variable is not free in the set of other assumptions and in the goal (see [3]).