A Goal-Based Approach to Guide the Design of an Abstract Event-B Specification

Abderrahman Matoussi, Frédéric Gervais, Régine Laleau

A Goal-Based Approach to Guide the Design of an Abstract Event-B Specification

Abderrahman Matoussi, Frédéric Gervais and Régine Laleau
Université Paris-Est, LACL, IUT Sénart Fontainebleau,
Dpt Informatique, Route Hurtault, 77300 Fontainebleau, France
{*abderrahman.matoussi, frederic.gervais, laleau*}*@u-pec.fr*

***Abstract*—With most of formal methods, an initial formal model can be refined in multiple steps, until the final refinement contains enough details for an implementation. Most of the time, this initial model is built from the description obtained by the requirements analysis. Unfortunately, this transition from the requirements phase to the formal specification phase is one of the most painful steps and is still ambiguous. In fact, building this initial model requires a high level of competence and a lot of practice, especially as there is no well-defined process to assist designers. For that purpose, we propose a goal-based approach in which initial formal models (in Event-B) are built incrementally driven by a goal-oriented requirements engineering (GORE) paradigm.**

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

I. Introduction

Formal methods have shown their ability to produce and improve complex systems for large industrial problems such as Paris metro line 14 [1] or Roissy Val [2] using the B method [3]. However, a serious problem with formal methods is the difficulty of using them. In fact, even if the formal development chain from abstraction via refinement to implementation is well understood, the major remaining weakness in this chain is that there is no well-defined process to assist designers in the building of the initial formal specification. Most of the time, this initial model is built "intuitively" from the description obtained by the requirements analysis and it requires a high level of competence and a lot of practice. Therefore, it will be difficult to fully comprehend the correspondence between requirements and initial formal specifications, and the validation of these specifications is very difficult mainly due to: (i) the inability for stakeholders to understand formal models; (ii) the inability for designers to link them with the initial requirements. It can result that an initial formal model may not be a correct realization of the requirements.

In this paper, we explore how to cope with this problem using Goal Oriented Requirements Engineering (GORE) approach [4] and the Event-B formal method [5]. The main objective is that this combination helps software designers to elaborate pertinent abstract Event-B specifications. The proposed approach aims to build incrementally abstract Event-B models from GORE goal models. Applying Requirements Engineering (RE) methods at the very beginning of a design process and before using formal methods can be interesting since these methods provide a rich way of structuring and documenting the entire requirements documents. Furthermore, whereas specifications allow us to answer the question WHAT the system does, RE methods allow us to address the WHY, WHO, WHEN questions. Among RE methods, Goal Oriented Requirements Engineering (GORE) paradigm [4] is particularly well-suited for requirements engineering since it is nearer to the way human thinks and is easy to understand by all stakeholders. Moreover, it offers some benefits in terms of: (i) providing the rationale for requirements and explaining them to stakeholders; (ii) reasoning on the system boundaries; (iii) identifying the responsibilities.

This paper continues our previous work [6] with additional studies, results and proofs. It describes a work in the framework of a research project, called TACOS [7], aiming at combining requirements engineering methods with formal methods. The remainder of this paper 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 in building the architecture of an Event-B specification from a KAOS goal model. Sections 4, 5 and 6 present the Event-B semantics of, respectively, the milestone, the AND and the OR goal refinement patterns. Relevant issues and benefits are discussed in Section 7. Section 8 presents the tool support of the proposed approach. Related work are discussed in Section 9. We conclude our paper in Section 10 with an outline of future work.

II. Background

In this section, we briefly introduce KAOS and Event-B.

A. KAOS method

KAOS (Keep All Objectives Satisfied) [8] 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 where 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) 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 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". These two kinds of goals represent what are currently called, **functional goals**. *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 one of its subgoals. KAOS offers a lot of refinement patterns 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 [8] for a full description of these notions.

B. Event-B method

Event-B [5], an evolution of the classical B method [3], 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: (i) *the guard strengthening* 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. (ii) *the correct refinement* ensures that the concrete event transforms the concrete variables in a way which does not contradict the abstract event. Event-B is supported by several tools, currently in the form a platform called Rodin [9].

III. From goals to Event-B

A. Motivation

With a simple system with few requirements, there is no problem for specifying it. However, the task becomes difficult for a real-scale system, with a large number of requirements, and a documentation which is often too verbose. In such system, people have expressed frustration in trying to use formal methods. Many reasons have been suggested for this situation such as a claim that they require a high level of competence and a lot of practice, especially to design the initial specification. In such situation, people try to do some kind of pseudo-programming instead of building an initial formal model. This kind of design constitutes a serious shortcoming because people are unable after to ensure traceability with the requirements documents. So, the idea encouraged by many researchers like J.R Abrial in [5] is to completely rewrite requirements documents before starting

any modeling. For that, requirements engineering methods such GORE can be advantageous since they provide a rich way of structuring and documenting the entire requirements documents. These methods are nearer to the way human thinks and are easy to understand by all stakeholders. In this paper, we aim to explore GORE approach to guide software designers in the elaboration of a pertinent abstract formal specification. We have chosen KAOS as a GORE 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* [10] for example (the other famous goal-oriented RE), the emphasis was more on qualitative reasoning on soft goals. Then, KAOS is particularly suitable to be transformed to formal languages like Event-B. 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.

In RE methods, two kinds of requirements can be identified. Functional requirements specify the functions that the system-to-be must be able to perform and non-functional requirements capture properties or constraints under which the system-to-be must operate, such as performance, quality, security concerns. Generally existing RE methods build models where functional and non-functional requirements are closely related. However, it appears in practice that these two kinds of requirements do not evolve over time in the same way and that functional requirements are more stable than non-functional ones. This is why our proposed method is based on the definition of three models. The first one describes functional requirements and is the base for deriving the skeleton of an Event-B specification. The second one describes non-functional requirements. The last one describes the impacts of non-functional requirements on functional requirements and will serve thus in completing the skeleton of the Event-B specification. In this paper, we focus on the functional requirements model and the derivation of formal specifications.

B. Achieve Patterns

To achieve our objective, we propose to give for each KAOS functional goal refinement pattern, an Event-B refinement structure. As mentioned in Section II.A, a functional goal can be an *Achieve goal* or its dual variant (*Cease goal*) [5]. 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**.*

C. Transformation 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 transformation is to express each KAOS goal as an 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. Refinements are denoted by a bubble linking the parent goal with an arrow, and the child goals with regular lines.

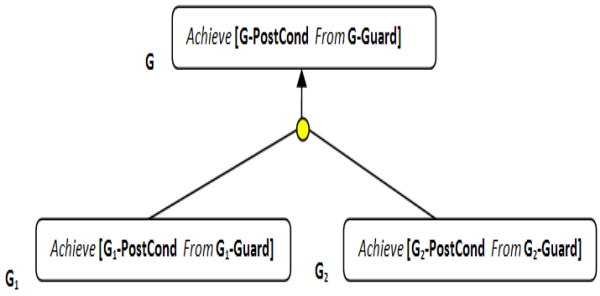

Figure 1. Example of KAOS goal model

Each level i ($i \in [1..n]$) 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 each goal as an 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).

Up to now, the initialization part, variables and invariants corresponding to their type, and contexts are manually completed by the designer. It would be possible to derive them from the object model in KAOS which contains every concept used in the definition of goals in the goal model. As this KAOS object model is an UML class diagram, the idea therefore is to reuse for example the UML-B work defined by [11], [12] that transforms a UML class diagram into a B machine and its associated context. Another important remark is that the invariants (that constraint the system) will be further added in the obtained Event-B machines when the non-functional goals (the KAOS *Maintain* goals) and their impacts on functional goals [13] will be translated in

MACHINE Abstract M_0	MACHINE First M_1 REFINES Abstract M_0
EvG $\triangleq$ **when** G-$Guard$ **then** G-$PostCond$ **end**	**EvG1** $\triangleq$ **when** G_1-$Guard$ **then** G_1-$PostCond$ **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 models obtained from the KAOS goal model

Event-B. This work is almost done and will be presented in a further work.

One may wonder how the temporal characteristic ("sooner or later" keyword) is expressed in our Event-B models, as this should imply to specify the concept of time. J.R Abrial [14] explains that the time dimension does 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 idea is shared by the authors of [15] when they describe the concept of "Time Bands". Therefore, this idea of time granularity 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 translated.

D. Example

Let us take a simple example of transformation extracted from [16], [17] that describe the Event-B specification of a localization software component using our proposed approach. This excerpt shows the Event-B counterpart of the high-level of the KAOS goal model related to a localization component. This high-level contains only one goal which is this strategic goal G, defined informally as follows:

Goal G: Achieve [LocalizeVehicle]

InformalDef: The vehicle must be localized.

We associate an Event-B model *Localization*, in Figure 3, to this most abstract level of the hierarchy of the KAOS goal graph. In this Event-B model, we will have an event

```
MACHINE Localization
SEES TypeSets
VARIABLES
        estimated_loc
INVARIANTS
        inv1 : estimated_loc ∈ LATITUDE × LONGITUDE
EVENTS
Initialisation
        begin
                act1 : estimated_loc := null ↦ null
        end
Event LocalizeVehicle ≙
        begin
                act1 : estimated_loc :∈ (LATITUDE \ {null}) ×
                        (LONGITUDE \ {null})
        end
END
```

Figure 3. An Example of the Event-B transformation

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 pair of latitude and longitude. At this level of abstraction, it is not necessary to precise the way this information is calculated. Thus, we use the non-deterministic generalized substitution through the symbol $:\in$ which specifies an unbounded choice. So, *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. Notice that at this abstraction level, the event **LocalizeVehicle** can always occur. Hence, its guard is always $true$. Sets in uppercase are abstract sets used to type the variables. They are described in the Event-B context *TypeSets* (see [16], [17]).

In the next sections, we propose an Event-B transformation rule for each KAOS refinement pattern associated to Achieve goals. Based on the classical set of inference rules from Event-B [5], 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) **Transformation to Event-B:** We propose here to transform the KAOS goal refinement pattern to Event-B as follows:
   a) **Formal definition:** We present the Event-B semantics given to each KAOS goal pattern.
   b) **Proof obligations identification:** We present just some informal arguments defining exactly what we have to prove for each KAOS goal refinement pattern. A formal argumentation of the identified proof obligations is detailed in [17].

IV. The milestone goal refinement pattern

A. Description of the KAOS pattern

The milestone goal refinement pattern [8] 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 4 (with just two sub-goals).

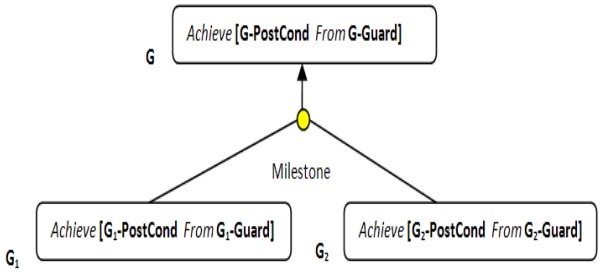

Figure 4. Milestone goal refinement pattern

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 stronger 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 stronger 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.

B. Transformation to Event-B

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**).

1) Formal definition: In Event-B, often the information about such events sequencing has to be embedded into guards and event actions with the downside of extra model variables. We present here another solution (without extra model variables) 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 **EvG** is refined as follows:

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

2) Proof obligations identification: We are going to give systematic rules defining exactly what we have to prove for this pattern in order to ensure that the sequence of concrete events **EvG1;EvG2** indeed refines its abstraction. In fact, in addition to *the feasibility proof obligation*[1], this kind of refinement requires to discharge these different proof obligations[2]:

- *The ordering constraint* expresses the "milestone" characteristic between the Event-B events.
$$G_1\text{-}PostCond \Rightarrow G_2\text{-}Guard \textbf{ (PO1)}$$
- *The guard strengthening* ensures that the concrete guard of the sequence (the guard of the first event in the sequence) implies the abstract guard.
$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \textbf{ (PO2)}$$
- *The correct refinement* ensures that the sequence (the action of the last event in the sequence) transforms the concrete variables in a way which does not contradict the abstract event.
$$G_2\text{-}PostCond \Rightarrow G\text{-}PostCond \textbf{ (PO3)}$$

V. The AND goal refinement pattern

A. 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).

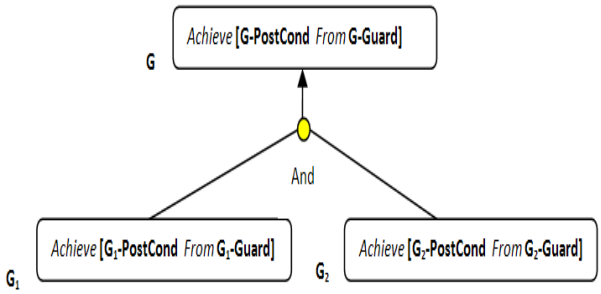

Figure 5. AND goal refinement

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 transformation to 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

[1]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.

[2]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.

strong solution is quite close to other solutions adopted by a lot of researchers such as J.R Abrial [3] 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.

B. Transformation to Event-B

As for the milestone goal refinement pattern, 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 [18]) 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.

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:

(**EvG1** $|||$ **EvG2**) Refines **EvG**

2) Proof obligations identification: We are going to give systematic rules defining exactly what we have to prove in order to ensure that the interleaving of concrete events indeed refines its abstraction. In fact, we have to prove three different lemmas (of course in addition to *the feasibility proof obligation*):

- *The guard strengthening* 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). Consequently, we have to prove that:

$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \textbf{ (PO1)}$$
$$G_2\text{-}Guard \Rightarrow G\text{-}Guard \textbf{ (PO2)}$$

- *The correct refinement* ensures that the interleaving of concrete events **EvG1** $|||$ **EvG2** transforms the concrete variables in a way which does not contradict the abstract event.

$$(G_1\text{-}PostCond \wedge G_2\text{-}PostCond) \Rightarrow G\text{-}PostCond \textbf{ (PO3)}$$

VI. The OR goal refinement pattern

A. 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).

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 introduces a certain kind of bounded non-determinism that will be resolved further in the implementation phase.

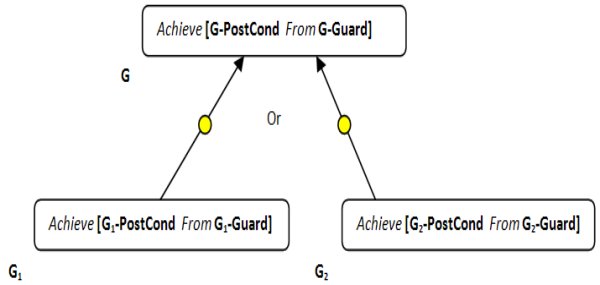

Figure 6. OR goal refinement pattern

B. Transformation to Event-B

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:

(**EvG1** XOR **EvG2**) Refines **EvG**

2) Proof obligations identification: We are going to give systematic rules defining exactly what we have to prove in order to ensure that each concrete event (**EvG1** or **EvG2**) indeed refines its abstraction.

- *The guard strengthening* ensures that the concrete guard (G_1-$Guard$ or G_2-$Guard$) is stronger than the abstract guard of **EvG**.

$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \textbf{ (PO1)}$$
$$G_2\text{-}Guard \Rightarrow G\text{-}Guard \textbf{ (PO2)}$$

- *The correct refinement* ensures that each concrete event (**EvG1** or **EvG2**) transforms the concrete variables in a way which does not contradict the abstract event **EvG**.

$$G_1\text{-}PostCond \Rightarrow G\text{-}PostCond \textbf{ (PO3)}$$
$$G_2\text{-}PostCond \Rightarrow G\text{-}PostCond \textbf{ (PO4)}$$

- *The "exclusive" characteristic* ensures that only one event (either **EvG1** or **EvG2**) but not both can be executed. Hence, the execution of one event forbids the other event to be fired.

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

Most of these proof obligations could be discharged by the current version of the Rodin automatic theorem prover [9]. 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**.

VII. Discussion and Benefits

One may wonder whether the transformation 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. However,

when deriving an Event-B specification from a goal model containing refinement following the milestone pattern that is interfered with other refinement patterns (see [19]), temporal ordering constraints between events need to be taken into account and have to be therefore necessarily and explicitly contained in the specification of events, for example in the guard. Then, model-checking techniques can be used. More generally, we use model-checking techniques to prove properties that contain temporal operators. Hence, our idea is to mix local proof obligations (generated by the Event-B prover) with proofs generated by a model-checker like ProB [20]. Recently, an interesting work-in-progress [21] presents an extension of Event-B with a mechanism to reason about event ordering based only on theorem proving and consequently it avoids state number explosion problem caused by model-checking. A number of proof obligations are generated when flows are attached to an Event-B model. The discharging of these proof obligations demonstrates the unfailing progress of a model through the events of a flow expression. Moreover, the author of [21] affirms that this control flow technique may also help to ensure a deadlock freeness, liveness and reachability properties of an Event-B model. Therefore, it would be benefit for us to explore this technique in order to verify the temporal ordering constraints between events without using model-checking techniques.

The proposed goal-based approach (KAOS) can guide software designers in the elaboration of pertinent abstract formal specifications since: (i) the visual dimension of KAOS (boxes, arrows...) allows people to better understand requirements and consequently to better design a faithful formal specification; (ii) GORE helps designers to structure an Event-B specification and to choose the right moment to introduce the different Event-B events; (iii) the usage of GORE methods help designers to correctly built guards of each Event-B event; (iv) finally, GORE methods encourage designers to consider both the system and its environment. This is an important point and this is exactly the aim of Event-B.

Once the abstract Event-B specification has been obtained from the whole KAOS goal model, the design of the software-to-be can begin. In other words, the B refinement process toward an implementation can begin. An interesting result is that the obtained abstract Event-B specification describes the properties that the final program must fulfill; i.e. it describes the way by which we can eventually judge that the final program is correct. This creates a better confidence in the development process and can help people who wants to certificate their product or software.

Another interesting result is that the obtained Event-B specification allows to give KAOS goal model a precise semantics just as existing translations from UML specifications into B specifications give a formal semantics to class diagrams or state diagrams [11], [12]. This point offers the possibility to prove some properties of consistency on the goal model by discharging the proof obligations derived 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 event corresponds exactly to the expression of the related goal since this latter is informal. For that, we can use an animation technique to validate the derived formal specification 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 [19] for more details. For that, ProB [20] may be a very useful validation tool since its automated animation facilities allow users to animate their specifications; i.e. gain confidence in their specifications.

VIII. Tool Support

An implementation, called SysKAOS2EventB plug-in, of the proposed approach has been elaborated using model-to-model transformation technologies. The developed tool is an initial step towards the development of a more complete tooling that allows the generation of an Event-B specification from requirements structured in the KAOS goal model. One of our priorities is that the tool supporting our approach must be open source developed using also others existing open source tools. However, the KAOS mother tool Objectiver[3] allowing the design of KAOS models is a proprietary tool. For that, an open source support tool [13] has been already developed based on the Topcased open platform [22] where the complete meta-model of KAOS goal model is defined as an extension of SysML [23]. This results on a new SysML profile, called SysML/KAOS, where all KAOS goal model concepts are integrated with the SysML requirements concepts. Topcased [22] is an open source software toolkit project integrated into Eclipse IDE. It is part of the MDE (Model Driven Engineering) process, which consists in developing tools on the basis of models. The advantage of Topcased is that it allows the creation of a proprietary meta-model editor using the EMF technology [24]. Once the SysML/KAOS goal diagram (the .sys diagram) is created, SysKAOS2EventB plug-in builds a Rodin project from this goal diagram (precisely from its .xmi counterpart). This is available by a right click on the project explorer of Topcased platform, pointing a SysML/KAOS goal model. SysKAOS2EventB plug-in creates a new Rodin project (with different MACHINE components) obtained

[3]http://www.objectiver.com/

by applying the transformation rules defined in this paper. SysKAOS2EventB plug-in was developed in Eclipse using a model-driven approach. It is based on the Event-B EMF framework [25] that provide an EMF based front-end to the Rodin platform [9]. As shown in Figure 7, SysKAOS2EventB development approach includes two main steps:

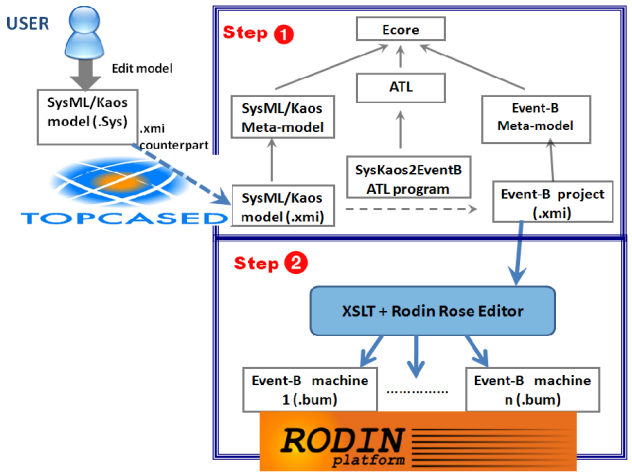

Figure 7. An overview of the SysKAOS2EventB plug-in

1) The objective of the first step is to transform the source xmi file (related to the SysML/KAOS goal diagram) to a target xmi file (related to the Rodin project). To this end, we have chosen ATLAS Transformation Language (ATL) [26] which is a language and a software environment well-suited for programming various transformations on trees/terms. An ATL transformation program defines how source elements are matched to target elements using both declarative and imperative rules. Both input and output meta-models of the ATL transformation are expressed in .ecore files which is a meta-model defined in the Eclipse EMF framework [24]. For our plug-in, we use two meta-models: one for SysML/KAOS language [23] and the second one for the Event-B language [25]. In fact, our program takes as input a KAOS model conform to the SysML/KAOS meta-model and produces an Event-B project conform to the Event-B meta-model.
2) The obtained xmi file (that contains the Event-B project) cannot be directly opened by Rodin platform. For that, the second step consists in performing a simple XSLT transformation[4] in order to obtain multiple xmi files where each one corresponds to one Event-B model. Each obtained xmi file is then loaded in Rodin [9] as a .bum file, thanks to the Rose Editor plug-in [25].

[4]http://www.w3.org/TR/xslt20/

Since the Rodin platform [9] will be used as input of the obtained Rodin project, we can now explore all the features offered by Rodin such, the prover or the ProB model-checker and animator [20]. SysKAOS2EventB plug-in is an initial step towards the development of a complete tool for narrowing the gap between the requirements analysis phase and the specification phase. This plug-in have enabled us to experiment different case studies. These experimentations confirm that GORE methods provide a possible way of building and structuring formal specifications.

IX. Related work

Our proposed approach aims at obtaining an abstract Event-B specification driven by KAOS goal models. 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.

KAOS provides an optional formal assertion layer for the specification of goals in Real-Time Linear Temporal Logic (RT-LTL). This formalization step [8], [27] allows to verify formally the completeness, minimality and consistency of goal refinements. The completeness and minimality conditions for instance can be checked via model checking [28]. 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. Consequently, it is difficult to validate specifications with regard to requirements even if they have been expressed with RT-LTL. Moreover, our proposed approach can be seen as an Event-B formalization of KAOS goal model based on a proof technique. Therefore it does not suffer from the state number explosion problem occurring in classical model checking on which [8], [27] is based.

The authors of [29] 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.

The work of [30] associates a B machine to each KAOS agent since agents are the active entities able to perform operations. For that, all the KAOS operations that an agent has to perform are represented by B operations. Moreover, all *Maintain goals* under the agent responsibility are translated as invariants of the corresponding B machine. We can also point out a work [31] 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 [32] 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, [33] 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 [34]. The source requirements are included as verification assertions that can be model-checked by tools like ProB [20], 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. Yet, non-operational goals play an important role for requirements completeness and pertinence and provide for example the rationale for the requirements that operationalize them. In this paper, we have explored how to cope with this problem using an approach that transform the whole KAOS goal model to abstract Event-B models. Our approach can be considered as complementary to existing ones.

X. Conclusion and further work

This paper presents an attempt to couple goal-oriented requirements specification with formal design specification to guide the software development in a constructive and provable way. This constructive approach driven by goals shows that it is possible to build abstract Event-B models driven by a GORE approach. Furthermore, what we present can be very useful in practice to systematically verify that all KAOS requirements are represented in the Event-B model. The reader can refer to [16], [17] that describe in details the application of our proposed approach in the framework of a research project, called TACOS [7]. We shared the salient points of our experience to specify a localization software component that used GPS, Wifi and sensors technologies. A number of future work are ongoing. We currently improve the developed meta-model [23] to represent non-functional goals and their impacts on functional goals which is inspired from the i* method [10] and from [35]. These elements will be considered as a sugar that enriches the obtained abstract Event-B models (new variables, new invariants...). We plan also to improve the presented tool support, SysKAOS2EventB plug-in, in order to consider these elements. Moreover, it would be interesting to establish the correspondence between the obtained abstract Event-B specification and the later phases of an Event-B development.

Acknowledgment

The work in this paper is partially supported by the TACOS project [7] ANR-06-SETI-017 founded by the french ANR (National Research Agency).

References

[1] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier, "Météor: A Successful Application of B in a Large Project," in *World Congress on Formal Methods*, ser. LNCS, J. M. Wing, J. Woodcock, and J. Davies, Eds., vol. 1708. Springer, 1999, pp. 369–387.

[2] F. Badeau and A. Amelot, "Using B as a High Level Programming Language in an Industrial Project: Roissy VAL," in *ZB*, ser. LNCS, H. Treharne, S. King, M. C. Henson, and S. A. Schneider, Eds., vol. 3455. Springer, 2005, pp. 334–354.

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

[4] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 35–46.

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

[6] A. Matoussi, F. Gervais, and R. Laleau, "A First Attempt to Express KAOS Refinement Patterns with Event B," in *ABZ'08*, ser. LNCS, E. Börger, M. J. Butler, J. P. Bowen, and P. Boca, Eds., vol. 5238. Springer, 2008, p. 338.

[7] TACOS Project: ANR-06-SETIN-017. [Online]. Available: http://tacos.loria.fr

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

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

[10] E. Yu, "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering," in *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, ser. RE '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 226–235.

[11] A. Mammar and R. Laleau, "A Formal Approach Based on UML and B for the Specification and Development of Database Applications," *Automated Software Engg.*, vol. 13, pp. 497–528, October 2006.

[12] C. Snook and M. Butler, "UML-B: Formal modeling and design aided by UML," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, pp. 92–122, January 2006.

[13] C. Gnaho and F. Semmak, "Une extension SysML pour l'ingénierie des exigences dirigée par les buts," in *INFORSID'10*, 2010, pp. 277–292.

[14] J. R. Abrial, "Extending B without changing it (for developing distributed systems)," in *H. Habrias editor, The First Conference on the B-Method*, November 1996, pp. 169–190.

[15] A. Burns and I. J. Hayes, "A Timeband Framework for Modelling Real-Time Systems," *Real-Time Syst.*, vol. 45, pp. 106–142, June 2010.

[16] A. Matoussi, F. Gervais, and R. Laleau, "Specification of a Localization Component Driven by a Goal-Based Approach: Some Lessons We Learned," in *SBMF'10*, ser. LNCS, J. Davies, L. Silva, and A. Simao, Eds., vol. 6527. Springer, 2011, pp. 177–193.

[17] ——, "An Event-B formalization of KAOS goal refinement patterns," LACL, University of Paris-Est, Tech. Rep. TR-LACL-2010-1, 2010. [Online]. Available: http://lacl.univ-paris12.fr/Rapports/TR/TR-LACL-2010-1.pdf

[18] D. Sangiorgi, "Locality and interleaving semantics in calculi for mobile processes," *Theor. Comput. Sci.*, vol. 155, pp. 39–83, February 1996.

[19] A. Mashkoor and A. Matoussi, "Towards Validation of Requirements Models," in *ABZ'10*, ser. LNCS, M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, Eds., vol. 5977. Springer, 2010, p. 404.

[20] M. Leuschel and M. J. Butler, "ProB: A Model Checker for B," in *FME*, ser. LNCS, K. Araki, S. Gnesi, and D. Mandrioli, Eds., vol. 2805. Springer, 2003, pp. 855–874.

[21] A. Iliasov, "On Event-B and Control Flow," DEPLOY Project, Tech. Rep., 2009. [Online]. Available: http://deploy-eprints.ecs.soton.ac.uk/144/

[22] TOPCASED - Toolkit in Open Source for Critical Applications Systems Development. [Online]. Available: http://www.topcased.org/

[23] R. Laleau, F. Semmak, A. Matoussi, D. Petit, A. Hammad, and B. Tatibouët, "A First Attempt to Combine SysML Requirements Diagrams and B," *ISSE*, vol. 6, no. 1-2, pp. 47–54, 2010.

[24] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed. Addison-Wesley, 2009.

[25] Event-B and Rodin Documentation Wiki. [Online]. Available: http://wiki.event-b.org/index.php/

[26] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. E. Rougui, "First experiments with the ATL model transformation language: Transforming XSLT into XQuery," in *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

[27] R. Darimont and A. van Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," in *Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, ser. SIGSOFT '96. New York, NY, USA: ACM, 1996, pp. 179–190.

[28] C. Ponsard, P. Massonet, J. F. Molderez, A. Rifaut, A. V. Lamsweerde, and H. T. Van, "Early verification and validation of mission critical systems," *Form. Methods Syst. Des.*, vol. 30, pp. 233–247, June 2007.

[29] R. Hassan, S. A. Bohner, S. El-Kassas, and M. Eltoweissy, "Goal-Oriented, B-Based Formal Derivation of Security Design Specifications from Security Requirements," in *ARES*. IEEE Computer Society, 2008, pp. 1443–1450.

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

[31] H. Nakagawa, K. Taguchi, and S. Honiden, "Formal specification generator for KAOS: model transformation approach to generate formal specifications from KAOS requirements models," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 531–532.

[32] I. El-Maddah and T. Maibaum, "Goal-Oriented Requirements Analysis for Process Control Systems Design," in *Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design*, ser. MEMOCODE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 45–46.

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

[34] J. Bicarregui, A. Arenas, B. Aziz, P. Massonet, and C. Ponsard, "Towards Modelling Obligations in Event-B," in *ABZ*, ser. LNCS, E. Börger, M. J. Butler, J. P. Bowen, and P. Boca, Eds., vol. 5238. Springer, 2008, pp. 181–194.

[35] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.