Bridging the gap between KAOS requirements models and B specifications

Abderrahman Matoussi, Régine Laleau, Dorian Petit

Bridging the gap between KAOS requirements models and B specifications

Abderrahman Matoussi **Régine Laleau** **Dorian Petit**

September 2009

TR–LACL–2009–5

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 Est)

Technical Report **TR–LACL–2009–5**

A. Matoussi, R. Laleau, D. Petit.
Bridging the gap between KAOS requirements models and B specifications

Bridging the gap between KAOS requirements models and B specifications

Abderrahman Matoussi[1] **Régine Laleau**[1] **Dorian Petit**[2]

[1] Laboratory of Algorithmics, Complexity and Logic - University Paris Est, France

{abderrahman.matoussi, laleau}@univ-paris12.fr

[2] University Lille Nord de France, F-59000 Lille,UVHC, LAMIH, F-59313 Valenciennes CNRS, UMR 8530, F-59313 Valenciennes, France

dorian.petit@univ-valenciennes.fr

Abstract

Employing formal methods for complex systems specification is steadily growing from year to year. Whereas the formal specification process from abstraction to implementation via refinement is well understood, the traceability between initial user requirements (requirements analysis) and the corresponding formal specification is still unsatisfying and ambiguous. In fact, there is little research on reconciling the requirements phase with the formal specification phase. Consequently, the gap between the requirements phase and the formal specification phase continues to grow larger and the reconciliation seems more and more difficult and complicated. Our objective is to combine these two phases by using KAOS and the B method. KAOS is a goal-oriented methodology for requirements engineering enabling analysis to build requirements models and to derive requirements documents. B is a model-based formal method supported by tools and that allows the design of systems, from specification to implementation. For that purpose, we propose to derive the architecture of the B specification from the KAOS goal model. This makes traceability between KAOS requirements and B models more explicit.

Keywords. Requirements engineering, KAOS methodology, B method, Traceability.

1 Introduction

Employing formal methods like B [1] and Z [2] 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 [20] or Roissy Val [21] using B method. With most of formal methods, an initial mathematical model is refined in multiple steps, until the final refinement contains enough details for an implementation. This initial model is derived from the user requirements (requirements analysis) which has to be carefully written before embarking in any computerized system development. As this formal development chain matures, the major remaining weakness in the development chain is the gap between textual or semi-formal requirements and the initial formal specification. 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). Consequently, the gap between the requirements phase and the formal specification phase gets larger and larger and the reconciliation seems more and more difficult. This report aims to bridge this gap using the requirements analysis method KAOS and the B formal method. For that, we propose a pragmatic approach to make traceability between requirements and B models more explicit since traceability has become a key subject of requirements engineering research [17]. In a first step, we do not attempt to automate traceability or to create a perfect mapping. Rather, we attempt to narrow the gap between the two phases by deriving the architecture of the formal specification. This allows systematic analysis, validation and verification of the mapping.

The remainder of the report is organized as follows. Section 2 overviews the KAOS and the B formal methods that are employed in the proposed approach. Section 3 details our traceability approach between a KAOS goal model and the B machines. Section 4 demonstrates the approach with a case study. Section 5 discusses related work. Finally, Section 6 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 B formal method and the KAOS method.

2.1 The B method

The B method [1] is a formal method for specifying, designing and coding software systems that integrates formal proof techniques in software development. This method share the same fundamentals as Z [2] and VDM [22]. It was used in the METEOR project [20], as well as in less well-known development projects [23, 24] and even non-critical development projects [25]. The software industry adopted B largely because of the availability of software tools supporting all phases of the B development process.

The B method is based on Abstract Machine Notation (AMN) and the use of formally proved refinements. Its mathematical basis is extracted from first-order logic, integer arithmetic and set theory. The original B (referred to as classical B, [1]) has a rich feature set that got consciously reduced in the more recent Event-B [3, 16] in order to make the notation easier to use and automated proving easier to implement. A B specification is structured in machines. One of the main parts of a B machine is the state space definition, which appears in the **VARIABLES** and **INVARIANT** clauses. The former enumerates the state components, and the latter defines restrictions on the possible values they can take. Operations allow machine variables to be transformed. The machine operations are defined by generalized substitutions, and each one contains: (i) a precondition represented by a predicate which expresses conditions to call this operation; (ii) an action represented by a substitution which expresses how machine variables evolve.
Machines in classical B are labeled according to their abstraction level: **MACHINE**, **REFINEMENT** or **IMPLEMENTATION**, from the most abstract to the most concrete:

- The abstract machine is the decomposition unit of the B method which contains: (i) *a static part* which specifies the system state. This specification defines the variables describing the state of system components and the invariants which are logical formulas

expressing the system static rules; (ii) *a dynamic part* which expresses the initialization and the evolution of the system state through a set of operations. Each operation must preserve the invariant of the machine.

- The refinement mechanism consists in reformulating, by successive steps, the variables and the operations of the abstract machine, so as to lead finally to a module which constitutes a running program. The intermediate steps of reformulation are called refinements and the last refinement level is called the implementation. A key merit of this refinement mechanism is the ability to preserve already proven system properties in higher level models. Hence, the refinement of an operation is correctly proven if it establishes the same result as its abstraction.

Each B component (**MACHINE**, **REFINEMENT** or **IMPLEMENTATION**) is structured using a single language, the B language. B models are accompanied by mathematical proofs, called proof obligations that guarantee correctness and effectiveness of system development. Industrial tools for the development of B based projects, which automatically generates proof obligations to be proven, have been available for a while now such as Atelier B[1].

2.2 KAOS methodology

KAOS (Knowledge Acquisition in autOmated Specification) [6, 7] is a goal-based requirements engineering method. KAOS requires the building of a data model in UML-like notation. The particularity of KAOS is that it is able to implement goal-based reasoning. 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 several 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. Goals are organized in a hierarchy obtained from the refinement of higher level goals into lower-level goals using the concept of refinement patterns [4]. 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** sums up all the behaviors that agents need to have to fulfill their requirements. Behaviors are expressed in term of operations performed by agents. Those operations work on objects described in the object model. So, they can create objects, provoke object state transitions or trigger other operations through send and receive events.

[1] http://www.atelierb.eu/

KAOS provides an optional formal assertion layer for the specification of goals in Real-Time Linear Temporal Logic (RT-LTL). Thanks to the use of developed formal reasoning techniques, analysts can identify and resolve conflicts between goals and prove absolute or partial goal satisfaction. A formal goal definition in KAOS begins with the description of the objects that the goal concerns. The definition then states the desired temporal ordering of states the concerned objects must hold in order to satisfy the goal. Goals are defined in the form: $C \Rightarrow op\ T$ where C and T are assertions about environmental situations, and op is a temporal operator that indicates the desired temporal nature of the target situation (T), in relation to a current situation (C).

KAOS provides a catalog of "Goal Patterns" that generalize the most common goal configurations. *Achieve Goals* ($C \Rightarrow \diamond T$) desire achievement "some time in the future". That is, the target must eventually occur. *Cease Goals* ($C \Rightarrow \diamond \neg T$) disallow achievement "some time in the future". That is, there must be a state in the future where the target does not occur. *Maintain Goals* ($C \Rightarrow T$) must hold "at all times in the future". *Avoid Goals* ($C \Rightarrow \neg T$) 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 sub-goals, such that the conjunction of the subgoals is a sufficient condition to achieve the parent goal. The OR-refinement relates 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 [4] 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. The sub-goals $G_1, G_2, ..., G_n (n \geq 2)$ refine a goal G in a domain theory Dom if the following conditions hold:

1. $G_1, G_2, \ldots, G_n, Dom \models G$ (entailment)
2. $\bigwedge_{j \neq i} G_j, Dom \not\models G$ for each $i \in [1..n]$ (minimality)
3. $G_1, G_2, \ldots, G_n, Dom \not\models false$ (consistency)

The first condition requires that the satisfaction of the subgoals together with the satisfaction of domain properties in Dom is sufficient for satisfying the parent goal G. The second condition requires that if a subgoal is left out of the refinement, the remaining subgoals are not sufficient for satisfying the parent goal. The third condition requires that the conjunction of the subgoals is logically consistent with the domain theory.

Thus, KAOS provides a set of domain independent refinement patterns which respect all these conditions and serve as guidance. These patterns help in improving the completeness and the consistency of goal models. They are grouped by the behavior prescription of the four high-level goals (Achieve, Avoid, Maintain and Cease) and can only be used in the context described by the tactics defined in KAOS.

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.

Unfortunately, the KAOS method stops at the requirements phase and doesn't address the other software development levels. This is a serious shortcoming since it obliges designers to use another method for developing their systems. Consequently, it is difficult to validate

specifications with regard to requirements. Nevertheless, contrary to other requirements methods such as i* [19], KAOS is promising in that it can be extended with an extra step of formality which can fill in the gap between requirements and the later phases of development. For instance, the fact that both KAOS and B employ first-order predicate logic facilitates the correspondence between KAOS requirements models and B specifications.

3 Correspondence rules between KAOS goal models and B specifications

The transition from the requirements phase to the formal specification phase is one of the most difficult steps in software development. The ideal solution would be to automatically derive B specifications from KAOS goal models. Unfortunately, this solution is very complicated and hard to set up since it is necessary to construct the body of B operations. To overcome these difficulties, we present a simpler solution that consists in defining a mapping between requirements and B models to improve traceability. There has been substantial research on traceability [17]. Commonly, pre- and post-traceability are distinguished. Traceability of requirements towards specification is called post-traceability (or forward traceability). Traceability of a requirement back to its origin is named pre-traceability (or backward traceability). In this report, we are interested in post-traceability only.

Since goals play an important role in requirements engineering process, the proposed mapping comes down to ensure traceability between KAOS goal models and B machines. Indeed, [18] affirms that goals provide a bridge linking stakeholder requests to system specification. As a consequence, rather than establishing traceability from the KAOS requirements model as a whole, we propose to establish traceability from individual goals that are part of the KAOS goal model.

The main idea is to build the architecture of the specification from the goal model. It consists in defining what a B machine contains and the links between the different machines. A B machine M_G is associated with each goal G. This machine M_G contains an operation called OP_G that operationalizes this goal G; i.e. it describes the "work" to perform to reach the goal G, in terms of generalized substitutions. Moreover, each refinement of the goal G is represented by a B refinement machine $RefM_G$ that refines the machine M_G; i.e. the abstract operation OP_G is refined by a concrete one, called $RefOP_G$ [2]. Therefore, we have explored the natural complementarities between KAOS and B because both of them have the notion of refinement and are based on a constructive approach of refinement. It means that the B refinement verifies the above-mentioned KAOS refinement conditions. Of course, this needs to be formally proved, this is an ongoing work.

The following issue consists in structuring the B specification. There are a number of ways to structure these B machines. In fact, B offers a lot of relationships, such as the inclusion mechanism. It is the main mechanism for structuring large machines. It allows the abstract system state to be divided into several independent parts, each encapsulated by a separate included machine. These parts can then be combined into a single "including" B machines using the INCLUDES clause.

Consequently, our idea is that the B machines $M_{G1}...M_{Gn}$ (related to the sub-goals of the goal G) are linked to the B machine $RefM_G$ via the inclusion relationship INCLUDES. As the including machine state $RefM_G$ is the combined state of all its included machines, the

[2] Usually, in B, the refined operation has the same name as the abstract operation

operation $RefOP_G$ which changes the state is built by combining operations $(OP_{G1}...OP_{Gn})$ of the included machines $M_{G1}...M_{Gn}$. The nature of this combination depends on the goal refinement pattern. We can distinguish three cases:

- *Milestone goal refinement pattern*: It prescribes that some milestone states are mandatory in order to reach a final one. For that, we consider that all the invoked operations $(OP_{G1}...OP_{Gn})$ are structured as a sequence of operations in $RefOP_G$, separated by the B sequential operator over substitutions (;) as shown in Figure 1.

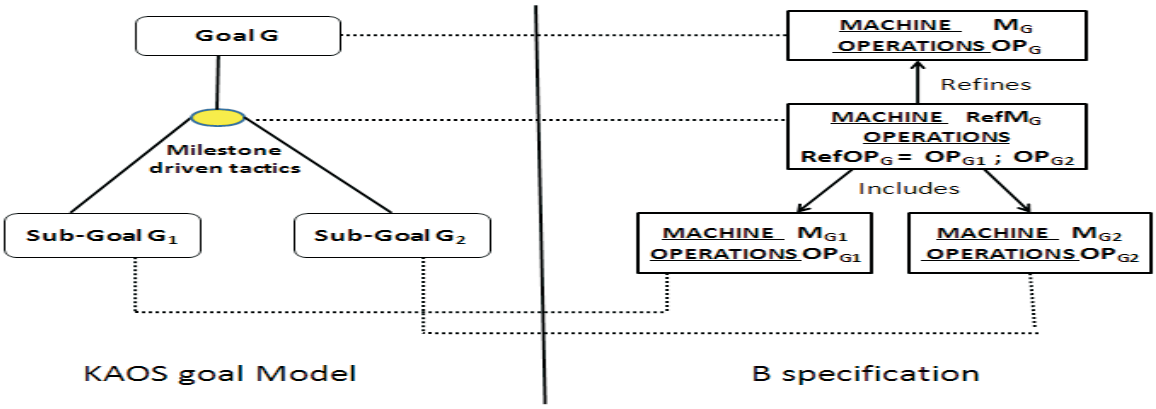

Figure 1: Traceability associated to the milestone goal refinement

- *Basic AND goal refinement*: It states that the parent goal G is satisfied if all the sub-goals are satisfied. For that, our idea is that the operation $RefOP_G$ invokes the operations $(OP_{G1}...OP_{Gn})$ and executes them at the same time. Therefore, we consider that all these operations are synchronously composed through the B parallel operator over substitutions (||) as shown in Figure 2.

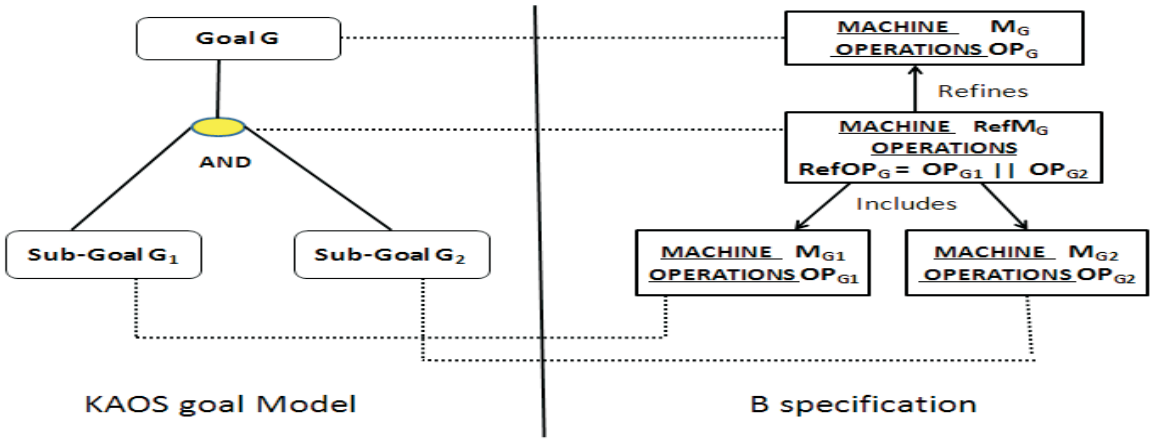

Figure 2: Traceability associated to the AND goal refinement

- *OR goal refinement*: It capture alternative ways of achieving goal G. Hence, we consider that the invoked operations $(OP_{G1}...OP_{Gn})$ are executed in a nondeterministic way in

$RefOP_G$, separated by the B operator ($CHOICE$) as shown in Figure 3. In fact, this B operator introduces a certain kind of bounded non-determinism.

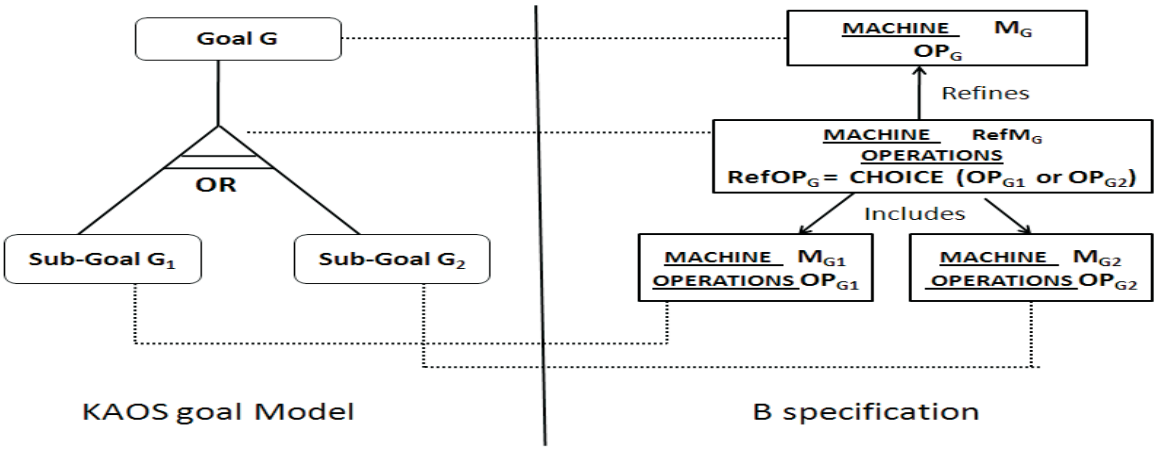

Figure 3: Traceability associated to the OR goal refinement

Combining all the above-mentioned rules allows us to obtain a architecture of a B model. Furthermore, the obtained architecture allows us to easily established traceability links between KAOS goals and the B operations that operationalize these goals.

4 Case Study: The localization component

The objective of this section is to demonstrate the proposed approach through the localization component case study. The coming subsections briefly describes the localization component requirements followed by showing how to elaborate these requirements in KAOS and how to ensure traceability between KAOS models and B specifications.

4.1 An overview of the localization component

A localization system 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.

Services relying on GALILEO will benefit from a better accuracy estimated to less than 2 meters against 5 to 10 meters for the GPS system. One of the novel aspects of GALILEO resides in a new integrity signal to inform users whether they can trust or not the signal transmitted to them. The objective is here to improve the confidence in the positionning information to envisage the deployment of more critical applications.

Positioning systems are often dedicated to a particular environment; the GNSS technology, for example, generally does not work indoors. To resolve these problems, numerous alternative relying on very different technologies have arisen. In the last few 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 netowrks such as Wifi or Bluetooth [26, 27], sensors [28], GNSS repeaters or visual landmarks.

4.2 The KAOS goal model of the localization component

In this report we have modelize a localization component based on several basic and off-the-shelf technologies : GPS, WIFI and sensors. This localization system will be used in a model of an autonomous vehicule : a CyCab. Figure 4 show a KAOS goal model of a localization component. Each Goal is described informally in natural language.

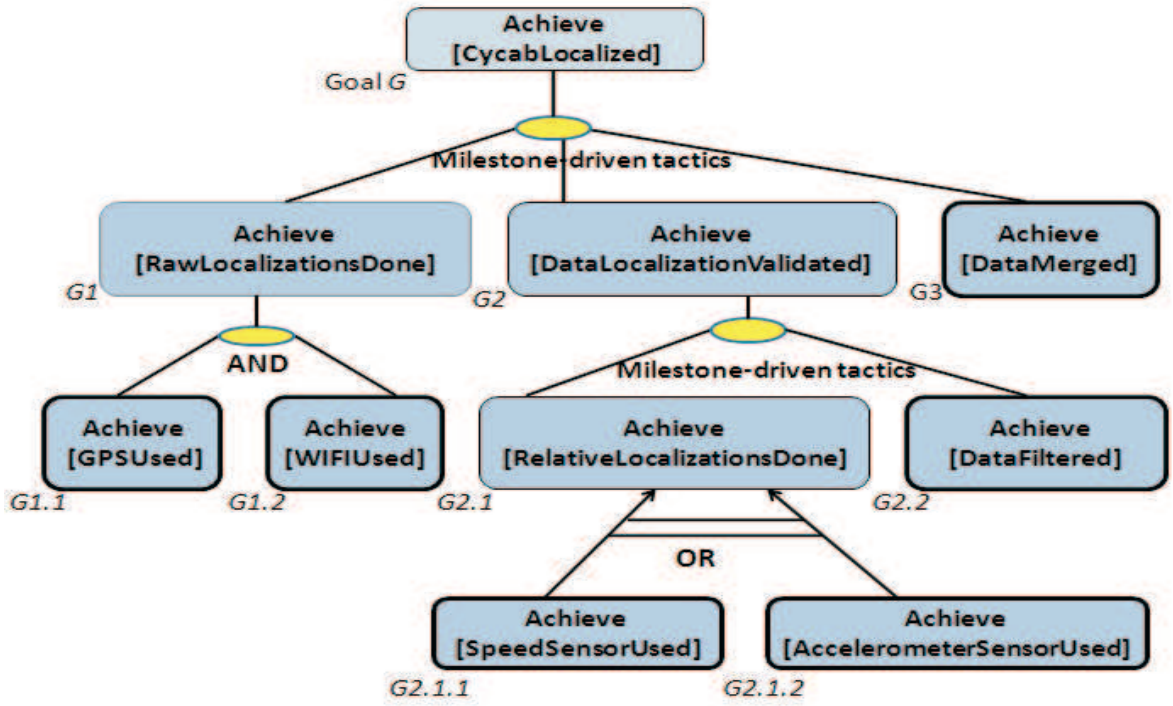

Figure 4: KAOS goal model of a localization component

In this report, we focus on the most frequently used "Goal Patterns" : the *Achieve goals*. The other "Goals Patterns" will be studied in further works. The high level goal G is defined as follows:

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

In order to build the architecture of the specification from the above KAOS goal model, we must define what a B machine contains and the links between the different obtained B machines. As explained in Section 3, a B machine is associated with each goal. For instance, we associate a B machine *Location* to the most abstract goal G. In this B machine, we will have an operation called **locate** that will translate this goal; i.e. it describes the "work" to perform to reach the goal G, in terms of generalized substitutions. If we consider that obtaining a localization is to get a latitude and a longitude, we obtain the operation presented in Figure 5. Sets in upercase are abstract sets used to type the variables and are described in the B machine *Type_sets* as shown in Figure 6.

MACHINE
Location
SEES
Type_sets
OPERATIONS

estimated_location ← **locate** =
ANY *info_lat* , *info_long*
WHERE $info\_lat \in LATITUDE \wedge info\_long \in LONGITUDE$
THEN $estimated\_location := (info\_lat \mapsto info\_long)$
END

END

Figure 5: The B machine *Location* associated to the abstract goal G

MACHINE *Type_sets*
SETS
$SUBCOMPONENTS = \{ gps , wifi \}$;
$SUBSENSORS = \{ speed , accel \}$
CONCRETE_CONSTANTS
LATITUDE , *LONGITUDE* , *DISTANCE*
PROPERTIES
$LATITUDE = \mathbf{NAT}$
$\wedge\ LONGITUDE = \mathbf{NAT}$
$\wedge\ DISTANCE = \mathbf{NAT}$
END

Figure 6: The B machine *Type_sets* used to declare sets and constantes

4.2.1 First refinement.

The goal G is refined into three sub-goals according to the milestone-driven tactics:

Goal G_1: Achieve [RawLocalizationsDone]
InformalDef: Firstly, a raw localization will be done.

Goal G_2: Achieve [DataLocalizationValidated]
InformalDef: Then, all the localization data will be validated.

Goal G_3: Achieve [DataMerged]
InformalDef: Finally, all the data will be merged in order to obtain the final localization.

Similarly, the sub-goals G_1, G_2 and G_3 are represented by three B machines *RawLocation*, $Validation$ and $Fusion$ (see Figures 7, 8 and 9), respectively. In each B machine, we will have a B operation that will translate these goals: **locate-raw**, **validate** and **merge** (respectively).

MACHINE
Raw_location
SEES
Type_sets
OPERATIONS

subcomponents_locations ← **locate_raw** =
ANY *sc_locations*
WHERE $sc\_locations \in SUBCOMPONENTS \rightarrow (LATITUDE \times LONGITUDE)$
THEN *subcomponents_locations* := *sc_locations*
END

END

Figure 7: The B machine *Raw-Location* associated to the goal $G1$

MACHINE
Validation
SEES
Type_sets
OPERATIONS

validated_locations ← **validate** (*raw_locations*) =
PRE

$raw\_locations \in SUBCOMPONENTS \rightarrow (LATITUDE \times LONGITUDE)$
THEN
ANY *valid_locations*
WHERE $valid\_locations \subseteq raw\_locations$
THEN *validated_locations* := *valid_locations*
END
END

END

Figure 8: The B machine $Validation$ associated to the goal $G2$

MACHINE
Fusion
SEES
Type_sets
OPERATIONS

merged_location ← **merge** (*raw_locations*) =
PRE
$raw\_locations \in SUBCOMPONENTS \nrightarrow (LATITUDE \times LONGITUDE)$
THEN
ANY *estimate*
WHERE $estimate \in LATITUDE \times LONGITUDE$
THEN *merged_location* := *estimate*
END
END

END

Figure 9: The B machine $Fusion$ associated to the goal $G3$

```
REFINEMENT
Location_r
REFINES
Location
SEES
Type_sets
INCLUDES
  Raw_location , Validation , Fusion
OPERATIONS

  estimated_location ← locate =
    VAR subcomponents_locations , validated_locations
    IN
      subcomponents_locations ← locate_raw ;
      validated_locations ← validate (subcomponents_locations ) ;
      estimated_location ← merge (validated_locations )
  END

END
```

Figure 10: The refinement machine *Location_r*

Since the goal G is refined into three sub-goals G_1, G_2 and G_3 according to the milestone goal refinement pattern, the abstract B machine *Location* is refined by the refinement machine *Location_r*. Consequently, the abstract operation **locate** is refined by a concrete one, called also **locate** (presented in Figure 10). This latter is built by combining the operations (**locate-raw**, **validate** and **merge**). These invoked operations are structured as a sequence of operations in **locate-R**, separated by the B sequential operator over substitutions (;). For that, the refinement machine *Location_r* must include the three B machines *Raw-Location* , $Validation$ and $Fusion$ that contains the invoked operations.

4.2.2 Second refinement.

The goal G_1 is AND-refined in two subgoals:

Goal $G_{1.1}$: Achieve [GPSUsed]
InformalDef: The Cycab must use the GPS system.

Goal $G_{1.2}$: Achieve [WIFIUsed]
InformalDef: The Cycab must use the wireless technique.

Similarly, the sub-goals $G_{1.1}$, $G_{1.2}$ are represented by two B machines GPS and $WIFI$ (see Figures 11, 12), respectively. In each B machine, we will have a B operation that will translate these goals: **locate_gps** and **locate_wifi** (respectively).

The abstract B machine *Raw-Location* is refined by the refinement machine *Raw-Location_r*. Consequently, the abstract operation **locate-raw** is refined by a concrete one (presented in Figure 13). This latter invokes the operations (**locate_gps** and **locate_wifi**) and executes them at the same time through the B parallel operator over substitutions (||).

```
MACHINE
  GPS
SEES
  Type_sets

OPERATIONS
  gps_loc ← locate_gps =
    ANY info_lat , info_long
    WHERE info_lat ∈ LATITUDE ∧ info_long ∈ LONGITUDE
    THEN
      gps_loc := { (info_lat ↦ info_long ) }
    END

END
```

Figure 11: The B machine GPS associated to the abstract goal $G_{1.1}$

```
MACHINE
  WIFI
SEES
  Type_sets
OPERATIONS
    wifi_loc ← locate_wifi =
    ANY info_lat , info_long
    WHERE info_lat ∈ LATITUDE ∧ info_long ∈ LONGITUDE
    THEN wifi_loc := {( info_lat ↦ info_long )}
    END
END
```

Figure 12: The B machine $WIFI$ associated to the abstract goal $G_{1.2}$

```
REFINEMENT
  Raw_location_r
REFINES
  Raw_location
SEES
  Type_sets
INCLUDES
  GPS , WIFI
OPERATIONS

  subcomponents_locations ← locate_raw =
    VAR gps_loc , wifi_loc
    IN
      gps_loc ← locate_gps || wifi_loc ← locate_wifi ;
      subcomponents_locations := ( { gps } × gps_loc ) ∪ ( { wifi } × wifi_loc )
    END
END
```

Figure 13: The refinement machine *Raw-Location_r*

MACHINE
Rel_location
SEES
Type_sets
OPERATIONS

sensors_locations ← **locate_rel** =
ANY *sens_locations*
WHERE *sens_locations* ∈ *SUBSENSORS* → (*LATITUDE* × *LONGITUDE*)
THEN *sensors_locations* := *sens_locations*
END

END

Figure 14: The B machine *Rel_location* associated to the abstract goal $G_{2.1}$

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

Goal $_{2.1}$: Achieve [RelativeLocalizationsDone]
InformalDef: At first, a relative localization will be done.

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

The transformation into B machines (see Figures 14, 15 and 16) is done exactly as for the first refinement.

MACHINE
Filter
SEES
Type_sets
OPERATIONS

kept_locations ← **filter_locations** (*new_raw_locations* , *sensors_locations*) =
PRE
new_raw_locations ∈ *SUBCOMPONENTS* → (*LATITUDE* × *LONGITUDE*)
∧ *sensors_locations* ∈ *SUBSENSORS* → (*LATITUDE* × *LONGITUDE*)
THEN

ANY
k_locations
WHERE
k_locations ⊆ *new_raw_locations*
THEN
kept_locations := *k_locations*
END
END

END

Figure 15: The B machine *Filter* associated to the abstract goal $G_{2.2}$

```
REFINEMENT
Validation_r
REFINES
  Validation
SEES
  Type_sets
INCLUDES
  Rel_location , Filter
OPERATIONS
    validated_locations ← validate (raw_locations ) =
    VAR sensors_locations
    IN
      sensors_locations ← locate_rel ;
      ASSERT
        raw_locations ∈ SUBCOMPONENTS → (LATITUDE × LONGITUDE )
         ∧ sensors_locations ∈ SUBSENSORS → ( LATITUDE × LONGITUDE )
      THEN
        validated_locations ← filter_locations (raw_locations, sensors_locations)
        END
    END

END
```

Figure 16: The refinement machine $Validation\_r$

```
MACHINE
    Speed
SEES
    Type_sets
OPERATIONS
    speed_location ← locate_speed =
    ANY info_location
    WHERE info_location ∈ {(LATITUDE × LONGITUDE)}
    THEN
      speed_location := info_location
    END
END
```

Figure 17: The B machine *Speed* associated to the abstract goal $G_{2.1.1}$

4.2.3 Third refinement.

The goal $G_{2.1}$ is OR-refined in two subgoals:

Goal $G_{2.1.1}$: Achieve [SpeedSensorUsed]
InformalDef: The Cycab may use a sensor system.

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

As for the other refinement levels, the sub-goals $G_{2.1.1}$, $G_{2.1.2}$ are represented by two B machines *Speed* and *Accel* (see Figures 17, 18), respectively. In each B machine, we will have a B operation that will translate these goals: **locate_speed** and **locate_accel** (respectively).

The abstract B machine *Rel_location* is refined by the refinement machine $Rel\_location\_r$. Consequently, the abstract operation **locate_rel** is refined by a concrete one (presented

```
MACHINE
    Accel
SEES
    Type_sets

OPERATIONS
    accel_location ← locate_accel =
    ANY info_location
    WHERE info_location : (LATITUDE × LONGITUDE)
    THEN
      accel_location := info_location
    END
END
```

Figure 18: The B machine *Accel* associated to the abstract goal $G_{2.1.2}$

in Figure 19). This latter invokes the operations (**locate_accel** and **locate_speed**) and executes them in a nondeterministic way through the B parallel operator over substitutions ($CHOICE$).

```
REFINEMENT
Rel_location_r
REFINES
Rel_location
SEES
    Type_sets
INCLUDES
    Speed, Accel
OPERATIONS
    sensors_locations ← locate_rel =
    CHOICE
        VAR speed_loc
        IN
            speed_loc ← locate_speed ;
            sensors_locations := ( {speed} × speed_loc )
        END
    OR
        VAR accel_loc
        IN
            accel_loc ← locate_accel ;
            sensors_locations := ( {accel} × accel_loc )
        END
    END

END
```

Figure 19: The refinement machine *Rel_location_r*

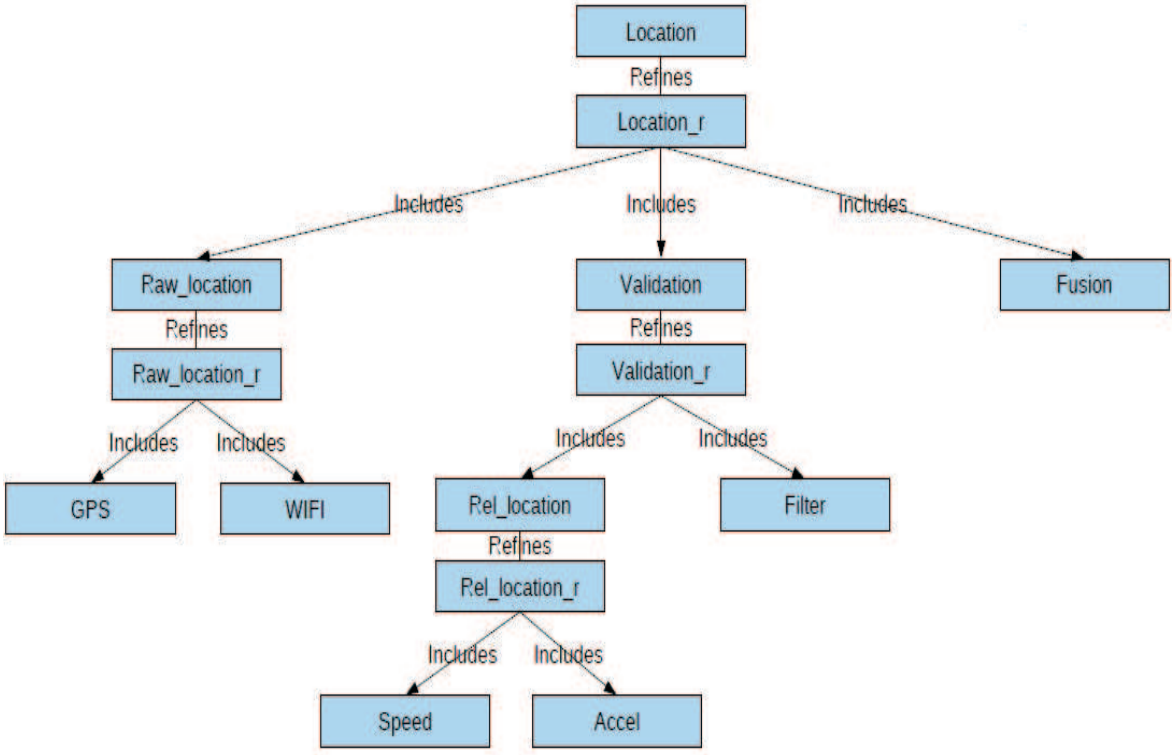

Figure 20: The architecture of the B machines

Figure 20 shows the structure of the different B machines. The obtained architecture allows us to easily established traceability links between KAOS goals and the B operations that operationalize these goals.

5 Related work

Our proposed approach aims at improving traceability between KAOS models and B specifications. In the sequel, we outline a number of approaches that have tried to bridge the gap between KAOS requirements model and formal methods.

An early attempt to bridge requirements to specification in the context of the B method is presented in [12] which proposes 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 [11] 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 [5] 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, [15] presents a constructive verification-based approach that try to bridge the gap between declarative requirements and operational system specifications in a rigorous manner. The approach consists in linking high-level system requirements, expressed as linear temporal logic formulae, to a system specification expressed as an Event-B machine extended with the notion of obligations [13]. The source requirements are included as verification assertions that can be model-checked by tools like ProB [14], showing that the proposed specification indeed meets the system requirements.

[10] presents very briefly a new approach that consists in including the requirements analysis phase in the software development associated with the formal methods by deriving an Event-B specification from a KAOS goal model. The interest of this constructive approach (driven by goals) is that the obtained specification preserves the properties of KAOS models. Thus, [10] shows that it is possible to express KAOS goal models with formal method like Event-B by staying at the same abstraction level.

Nevertheless, the traceability expressed by most 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 do not include any information about the non-operational goals and the type of goal refinement. In this report, we have explored how to cope with this problem using a new traceability approach between the whole KAOS goal model and the formal model. Hence, what we present can be very useful in practice to (i) systematically verify that all KAOS requirements are represented in the B model; (ii) systematically verify that each element in the B model has a purpose in KAOS.

6 Conclusion and further work

In this report, we present a mapping between KAOS requirements and B models. The main contribution of our approach is that it establishes a bridge between the non-formal and the formal worlds as narrow and concise as possible. Moreover, this bridge balances the tradeoff between complexity of rigid formality (B method) and expressiveness of semi-formal

approaches (KAOS). While this report introduces and demonstrates the approach, a number of future research steps are ongoing. In fact, the current mapping is still partial and we are working on its extensions. Further work will consist in applying the approach on a number of case studies in order to support non-functional goals. At tool level, we plan to develop a connector that establish a partial automated traceability between KAOS goal models and B specifications.

Acknowledgment

The work in this report is supported by the TACOS project ANR-06-SETI-017 founded by the french ANR (National Research Agency).

References

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

[2] J.M. Spivey. *Understanding Z*. Cambridge University Press, 1988.

[3] J.R. Abrial and L. Mussat. Introducing dynamic constraints in B. In *B'98*, volume 1393 of *LNCS*, pages 83–128, Montpellier, France, April 1998. Springer-Verlag.

[4] 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.

[5] 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.

[6] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *RE 2001*, pp. 249–263, Toronto, Canada, August 2001. IEEE Computer Society.

[7] E. Letier. Reasoning About Agents in Goal-Oriented Requirements Engineering. Phd Thesis, Université Catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, Mai 2001.*ftp://ftp.info.ucl.ac.be/pub/thesis/letier.pdf*

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

[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] 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.

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

[13] 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.

[14] M. Leuschel and M. 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.

[15] 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.

[16] D. Cansell and D. Mery. Tutorial on the event-based B method : Concepts and case studies. In *FORTE 2006*, Paris, September 2006.

[17] OCZ. Gotel and CW Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, 1994. IEEE Computer Society Press.

[18] 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.

[19] E. Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 226-235, 1997. IEEE Computer Society Press.

[20] 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.

[21] 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 (ZB05)*, pages 334–354, Guildford, UK, 2005. Springer.

[22] D. Bjorner, and C. Jones. The Vienna Development Method: The Meta-Language 1978. Springer.

[23] M. Carnot, C. DaSilva, B. Dehbonei, and F. Mejia. Error-free software development for critical systems using the B-methodology. In *The Third IEEE International Symposium on Software Reliability Engineering*, pages 274–281, North Carolina, October 1992. IEEE Computer Society Press.

[24] B. Dehbonei and F. Mejia. Formal development of safety-critical software systems in railway signalling. In M. G. Hinchey and J. P. Bowen, editors, *Applications of Formal Methods*, Series in Computer Science, pages 227–252, 1995. Prentice Hall International.

[25] B. Tatibout, A. Requet, J.-C. Voisinet, and A. Hammad. Java card code generation from B specifications. In I. J. Dong and E. J. Woodcock, editors, *ICFEM*, volume 2885, pages 306–318. Formal Methods and Software Engineering, 2003. Springer.

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

[27] J.A. Royo and E. Mena and L.C. Gallego. Locating Users to Develop Location-Based Services in Wireless Local Area Networks. In *Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI2005)*, pages 471-478, Granada, Spain, 2005.

[28] 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