A Reverse-engineering Approach to Understanding B Specifications with UML Diagrams*

Akram Idani Yves Ledru Didier Bert

Université Joseph Fourier - CNRS - INPG
Laboratoire Logiciels, Systèmes, Réseaux - IMAG
B.P. 72 - F-38402 - Saint Martin d'Hères Cedex - France
{Akram.Idani, Yves.Ledru, Didier.Bert}@imag.fr

Abstract

Formal methods are nowadays the most rigorous way to produce software. However, the existing formal notations are not easy to use and understand for most people. Our approach proposes to circumvent this shortcoming by producing complementary graphical views on the formal developments. This paper addresses the graphical representation of formal B specifications using UML diagrams. A reverse-engineering approach is proposed to generate several class diagrams showing the static aspects of the B developments. These diagrams can help understand the specification for stakeholders who are not familiar with the B method, such as customers or certification authorities. A concept formation technique based on weighted link matrices is proposed to improve automation.

Keywords: *Concept formation, B, UML, Formal concept analysis.*

1. Introduction

While formal methods are focused on some particular parts of the systems, especially secure systems, graphical techniques are the most useful techniques to specify in a comprehensible way large and complex systems. The formal developments may be difficult to read because of the mathematical notations. This readability is important for the validation or the certification steps. On the one hand, formal notations ensure precision, and allow proofs and verification techniques but are not easy to read and understand. On the other hand, graphical notations produce specifications easy to read and understand but which can cause misinterpretations of user requirements because of the lack of a precise semantical basis.

In this paper we investigate a practical solution to assist the understanding of formal specifications written in B by providing complementary views of the specifications as UML class diagrams. This work starts from the fact that several significant formal developments are uniquely based on formal methods, for example the B method [1] has been used in industrial railway and smart card applications [3, 5]. And several companies, such as Siemens Transport [8] and Clearsy [19, 18], have an established software development process which follows the B method:

- B specifications are written from a requirements document or from a detailed analysis of the requirements. The consistency of these formal specifications is assessed with the support of a proof tool.
- These specifications follow a development process based on proved refinements that leads to executable programs.

Considering that the B method covers the entire development process starting from abstract specifications and leading to program code (C, Ada,...), we propose a reverse-engineering approach to produce complementary UML views on each refinement in order to ease understanding these specifications. Fig. 1 shows how our documentation activities are integrated with the overall B development process. The left hand side of the figure shows a classical B development. The B developer builds the initial B specification from the customer needs, and then constructs and proves a series of refinements until he reaches an implementation. On the right hand side of the figure, the reverse-engineering process presented in this paper builds UML views from the various formal documents produced during the development. They are intended to be read by the cus-

*This work is partially supported by the EDEMOI project of the French national ACI Sécurité informatique.

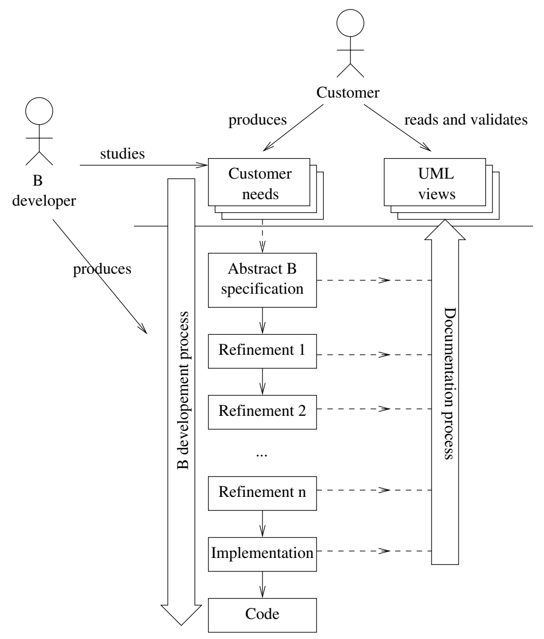


Figure 1. B/UML process

tomer, or by some certification authority, to validate that the B development fits the customer needs.

Early work in this direction includes [22, 9] which apply a set of rules to construct UML class diagrams as static views of B specifications. The major limit of these approaches is that they apply empirical rules or heuristics which could produce complex diagrams when the specifications scale up. We believe that applying alternate reverse-engineering methods addressing concept formation techniques is a useful and interesting contribution to the existing approaches. We developed a tool (B/UML tool [14]) which automatically produces several class diagrams from a B specification depending on some choices done by the user. The goal of this paper is to investigate metrics that will help the user choices and eventually provide a fully automatic mode to the tool.

This paper is organized as follows: in Section 2 we discuss the existing approaches which translate B specifications into graphical documentation. Section 3 gives an overview of the principles of our previous proposal. Section 4 introduces the notion of link matrix to encode the estimation of the pertinency of data in B specification with respect to the operation using them. In Section 5 we apply a concept formation technique based on weighted link matrices in order to improve the potential of automation of our previous approach. Finally, Section 6 draws the conclusions and perspectives of this work.

2. Graphical views of a simple B development

A significant effort has been devoted by the research community in order to establish links between UML and B. In particular, several approaches provide translations from annotated UML diagrams to B [21, 16, 15]. These approaches aim to take advantage of B tools while remaining integrated in a standard industrial process based on UML. In this paper we investigate the reverse approach: using graphical notations, such as UML diagrams, as a way to document formal B developments.

Graphical views can be built from a B specification using two kinds of tools:

i. Tools that represent the behaviour of the specifications. For example, the Pro-B tool [17] includes a model-checker which exhaustively explores the finite behaviour of the specification, and a graphical tool which displays the states and transitions covered by the model-checker. In [7] the authors propose a proof technique to explore all valid transitions of the system and use finite labeled transition systems to represent the behaviour of B specifications. In the same context, [11] proposes a set of derivation rules to generate state transition diagrams from B specifications. Finally, in [13], we present a tool that extracts abstract graphs from an exhaustive exploration of the behaviour and a set of abstraction predicates by combining proof and model-checking techniques.

ii. Tools that extract the static aspects of the B specifications. For example, [22] defines some rules to automatically derive an UML class diagram from formal B specifications. In the same context, [9] presents some heuristics which lead to construct interactively simpler diagrams. Finally, in [12, 14] we have presented a more automated approach to the derivation of class diagrams from formal B developments. The proposed approach is to apply a concept formation technique which takes into account operations of the B specifications.

Both kinds of tools are complementary. The first category helps to understand the behaviour of a specification while the second one focuses on the structure of its data. This paper discusses the second kind of tools and uses an introductive example to illustrate how the existing approaches [22, 9, 12, 14] translate a B specification into a class diagram.

2.1. The "*Access Control*" specification

The following specification taken from [2] describes the control of the access of persons to buildings. Set *PERSON* models the persons and set *BUILDING* models the buildings. Persons must have authorization to access a building. These authorizations are represented by a set of "person-building" pairs called aut. Variable sit links a given person to the building inside which he is currently. Set *DOOR* models the doors of the buildings. Each door is associated with a building of origin, represented by function $orig$ and a destination building, represented by function $dest$. Function unl links a person to the door which is currently opened for him (unlocked). The definition of $green$ gives the doors opened to some persons. Variable red means that a door is currently locked and no one can pass it. The definition of *admitted* states the condition for a person *pp* to ask for access through a door *dd*: the person must be situated in the building of origin of *dd*, it may not have unlocked another door, and must be authorized to access the destination building. The invariant also states that persons are only situated in authorized buildings, that a door is either red or green, that if a door is unlocked for a given person, then the person is situated in the origin building and has authorization for the destination.

```
MACHINE
    AccessControl
SETS
    BUILDING; PERSON; DOOR
CONSTANTS
    aut, orig, dest
PROPERTIES
    $aut \in PERSON \leftrightarrow BUILDING \wedge$
    $orig \in DOOR \rightarrow BUILDING \wedge$
    $dest \in DOOR \rightarrow BUILDING$
VARIABLES
    sit, unl, red
DEFINITIONS
    $green == \mathbf{ran}(unl)$ ;
    $admitted(pp,dd) == orig(dd) = sit(pp) \wedge$
            $pp \notin \mathbf{dom}(unl) \wedge (pp,dest(dd)) \in aut$
INVARIANT
    $sit \in PERSON \rightarrow BUILDING \wedge sit \subseteq aut$
    $unl \in PERSON \rightarrowtail\!\!\!\!\!\shortmid\;\; DOOR \wedge red \subseteq DOOR \wedge$
    $red \cap green = \emptyset \wedge (unl;orig) \subseteq sit \wedge (unl;dest) \subseteq aut$
```

The *AccessControl* specification features 5 operations which are :

- Operation *pass* represents the entrance of a person pp in a building *bb* for which he has the authorization.

```
pass =
    ANY pp, bb WHERE
        $pp \in PERSON \wedge bb \in BUILDING \wedge$
        $(pp,bb) \in aut \wedge sit(pp) \neq bb \wedge$
        $unl(pp) = dest^{-1}(bb)$
    THEN
        $sit(pp) := bb \;||$
        $unl := unl \rhd \{orig^{-1}(bb)\}$
    END;
```

- Operation *unlock* opens a door for a person who wishes to go from one building to another. This is done only if the person is allowed to go through the door.

```
unlock =
    ANY pp, qq WHERE
        $pp \in PERSON \wedge qq \in DOOR \wedge$
        $qq \notin (green \cup red) \wedge admitted(pp,qq)$
    THEN
        $unl := unl \cup \{pp \mapsto qq\}$
    END;
```

- Operation *refuse* locks the door of a building when a person is trying to go to the building without authorization $(\neg admitted(pp, qq))$.

```
refuse =
    ANY pp, qq WHERE
      $pp \in PERSON \wedge qq \in DOOR \wedge$
      $qq \notin (green \cup red) \wedge \neg\ (admitted(pp,qq))$
    THEN
      $red := red \cup \{qq\}$
    END;
```

- Operation *lock* closes a door.

```
lock =
    ANY qq WHERE
        $qq \in DOOR \wedge qq \in green$
    THEN
        $unl := unl \rhd \{qq\}$
    END;
```

- Operation *free* resets the red indicator of a locked door.

```
free =
    ANY qq WHERE
        $qq \in DOOR \wedge qq \in red$
    THEN
        $red := red - \{qq\}$
    END
```

2.2. Producing the graphical views from B specifications

Related works The state of the art includes the works of Tatibouet [22], Fekih [9] and our previous work on this topic [12, 14]. In Tatibouet's approach [22], classes are generated for every machine, set and relation of the B specification. Although it is an automatic approach to the generation of class diagrams, it results in complex diagrams with a large number of classes. Moreover, these classes are linked by numerous associations. For example, the abstract specification of *AccessControl* would result in 9 classes and 13 associations.

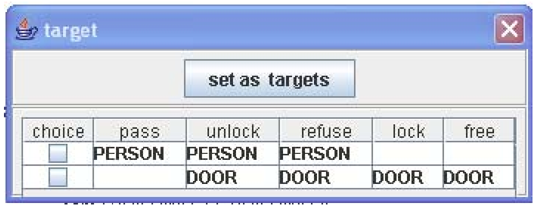
Rules given in [9] generate a class for every set which appears in the domain of a relation. For the *AccessControl* specification this leads to 3 classes corresponding to sets *PERSON*, *DOOR* and *BUILDING*. Although the generated diagrams are simpler than those generated by rules of

[22], the approach does not associate operations to classes. In fact, this step is done interactively by the user. As the *AccessControl* specification presents 5 operations, the approach of [9] results in $3^5 = 243$ possibilities to associate these operations to the 3 identified classes (e.g. *PERSON*, *DOOR* and *BUILDING*). This number easily explodes when specifications scale up. This clearly requires a tool support that is not really compatible with the informal character of the rules.

Our approach In [12, 14] we overcome this limit and keep the readability of the generated diagrams by identifying proper and shared operations. The B/UML tool developed in our laboratory associates a set of possible classes to each operation (these classes are called targets of operations). The user interacts with the tool once to choose a unique target for each operation. Then, the tool generates for each set of chosen targets a preliminary model called context model. This model is then automatically translated into a class diagram.

Fig. 2 shows the "target" window of our tool when applied to the *AccessControl* specifications. Two possible targets have been identified corresponding to sets *PERSON* and *DOOR*. We will show later how our tool identifies these targets. Each of the five operations must be associated to one of these candidates. The user simply clicks on the chosen target for each operation. He can also select the checkbox of a given line to set this class as target to all associated operations.

The tool identifies only 4 possible configurations corresponding to two user choices: each of operation *unlock* and *refuse* may be associated to either *PERSON* or *DOOR*. The other operations (*pass*, *lock* and *free*) may only be attached to one of the classes.



choice	pass	unlock	refuse	lock	free
☐	PERSON	PERSON	PERSON		
☐		DOOR	DOOR	DOOR	DOOR

Figure 2. The "target" window issued from the *AccessControl* specification

Operation *pass* may only be associated to class *PERSON* because *pass* deals only with persons and building. It is also the case for operations *lock* and *free* which are necessarily associated to class *DOOR*. These operations could not be methods of class *PERSON* because they acts only on doors to close and open them. Rules of [9] don't forbid such a wrong transformation. Our tool only offers to associate an operation with sets that it accesses or modifies.

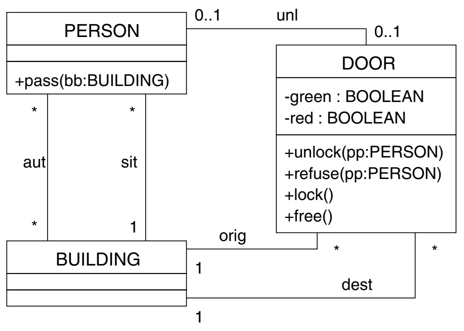


Figure 3. A generated class diagram from the *AccessControl* specifications

Fig. 3 shows the result of our tool when the user associates *unlock* and *refuse* to class *DOOR* rather than *PERSON*. In this figure *BUILDING* appears as a class without methods because it is not identified as a target of any operation.

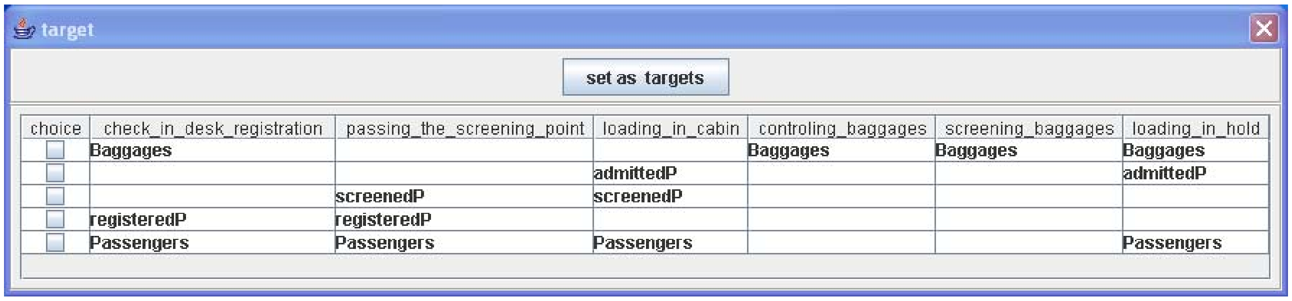
The major limitation of the tool is that it needs a small interaction with the user to choose a target class for each operation. Moreover, for more complex specifications such the *SecureFlight* specification[1] (222 source lines of B), the tool allows a greater number of targets. Fig. 4 displays the "target" window issued from the *SecureFlight* specifications and which gives a total of 81 possible configurations.

The tool allows 65 of the 81 configurations because they satisfy pertinency criteria [14]. In the following, we give an overview of these criterions and we propose a method to rank the choices of targets in order to help the user to identify the best configuration.

3. Contexts

The proposed approach is close to existing reverse-engineering approaches based on a concept formation technique such as [20, 23]. These approaches study the dependencies between operations and data (or concepts) of a source code in order to restructure it into an object oriented program. In our approach the term "**data concept**" inspired by [4] corresponds to data in the B specification (sets, constants and variables). We address the dependencies between data concepts and operations issued from a B specification by a data concept dependence relation $\mathcal{I}$ such that for a data concept d, $o \in \mathcal{I}[\{d\}]$ means that the operation o uses the data concept d. Three kinds of access can be identified: Modification, Reading and Precondition.

[1]The *SecureFlight* specification is taken from the EDEMOI project (http://www-lsr.imag.fr/EDEMOI/).



choice	check_in_desk_registration	passing_the_screening_point	loading_in_cabin	controling_baggages	screening_baggages	loading_in_hold
☐	Baggages			Baggages	Baggages	Baggages
☐			admittedP			admittedP
☐		screenedP	screenedP			
☐	registeredP	registeredP				
☐	Passengers	Passengers	Passengers			Passengers

Figure 4. The "target" window issued from the SecureFlight specification

A *Context* is a group of data concepts and B operations such that the data concepts are used by the operations. Similarly to classes in UML class diagrams, contexts are the basic entities of our preliminary model, they will be at the origin of classes. In fact, a class (respectively a context) groups a set of attributes (respectively a set of data concepts) and a set of methods (respectively a set of operations) that act on the attributes. The idea behind contexts is that they will be automatically translated into elements of a class diagram. Therefore, the major difficulty is to find the set of contexts which lead to pertinent classes in the UML class diagram.

Definition 1 *A **Context** is a pair $\mathcal{F} = (\mathcal{D}, \mathcal{O})$ where $\mathcal{D}$ is a set of data concepts and $\mathcal{O}$ a set of operations with:*

- $\mathcal{O} \subseteq \mathcal{I}[\mathcal{D}]$
- $\forall\, d \cdot (d \in \mathcal{D} \Rightarrow \exists\, o \cdot (o \in \mathcal{O} \wedge o \in \mathcal{I}[\{d\}]))$

Where $R[S]$ is the image of set S by relation R.

These properties mandate that all data concepts are used by the operations, and all operations use at least one data concept of the context. For example the following contexts correspond to the *AccessControl* specification:
$\mathcal{F}_1 = (\{$*DOOR*, *red*, *orig*,*unl*, *green*$\}, \{$*lock*, *free*, *unlock*, *refuse*$\})$
$\mathcal{F}_2 = (\{$*PERSON*, *sit*, *aut*, *BUILDING*$\}, \{$*pass*$\})$

A data concept d is said to be maximal if for any other data concept d' in the specification, the set of operations using d' is not included in the set of operations using d.

Definition 2 *A data concept $(d \in \mathcal{D})$ is maximal iff:*

$$\forall\, d' \cdot (d' \in \mathcal{D} - \{d\} \Rightarrow \mathcal{I}[\{d\}] \not\subset \mathcal{I}[\{d'\}])$$

We call $max(\mathcal{I})$ the set of all maximal data concepts of $\mathcal{I}$.

The data concept dependance relation issued from the *AccessControl* specifications gives 3 maximal data concepts which are: *PERSON*, *sit* and *DOOR*.

The maximal data concepts which are sets in the B specifications will be the classes of the class diagram. As only *PERSON* and *DOOR* are sets, they are considered as the best class candidates for the class diagram (For more details of the data concept maximality, refer to [12]).

The set $\mathcal{C}l$ of maximal data concepts which are considered as candidate classes present an interesting notion which is the notion of proper operations:

Definition 3 *Having a dependence relation $\mathcal{I}$ between a set of data concepts $\mathcal{D}$ and a set of operations $\mathcal{O}$ issued from a given B specification, and for a class concept d $(d \in \mathcal{C}l)$ we define the set of operations proper to d as:*

$$\mathcal{O}_{\mathcal{P}}(d) = \{o \mid \forall\, d' \cdot (d' \in \mathcal{C}l - \{d\} \Rightarrow o \in \mathcal{I}[\{d\}] - \mathcal{I}[\{d'\}])\}$$

i.e. an operation is proper if it is linked to a single class concept by $\mathcal{I}$.

Having:
$\mathcal{I}[\{PERSON\}] = \{pass, unlock, refuse\}$
$\mathcal{I}[\{DOOR\}] = \{lock, free, unlock, refuse\}$
we obtain:
$\mathcal{O}_{\mathcal{P}}(PERSON) = \{pass\}$
$\mathcal{O}_{\mathcal{P}}(DOOR) = \{lock, free\}$.

This section has introduced the basic notions of our approach (data concept, context, . . .). Details on how to form contexts and identify class concepts are given in [14]. This approach is supported by the B/UML tool. In the following section we propose a technique to improve our notion of pertinency and to rank the set of contexts formed by algorithm of [14].

4. Link matrix

The approach presented in [14] and our tool allow the user to associate any non-proper operation with any context linked to it.

The aim of the link matrices is to evaluate the pertinency of data concepts with respect to the context to which they are attached. Our study is based on the dependence between

B operations and a subset of data concepts of the B specification (sets, sub-sets and relations) in order to find a preliminary model which can be transformed automatically to a pertinent class diagram. The basic entities of the proposed model are contexts which are formed by analyzing the data concept dependence relation $\mathcal{I}$ issued from the specifications.

Considering a data concept d with $\mathcal{I}(d) = \{o_1, o_2, \ldots, o_n\}$ we define a matrix of the size 2×2^n. This matrix deals with binary valued propositions for the n operations which use d and specifies the fact that the data concept d is pertinent or not for each combination of these operations. For illustration we will consider data concept red which has three operations:

$$\mathcal{I}[\{red\}] = \{unlock, refuse, free\}$$

and the formal context:

$$\mathcal{F} = (\{..., red, ...\}, \{unlock, free\})$$

This means that red is grouped with two operations $unlock$ and $free$.

4.1. Boolean metrics

The simplest metrics to decide that the data concept red is pertinent or not with respect to the combinations of $unlock, refuse$ and $free$ is to compute an or-combination of the three operations. In other words, red is pertinent for a context $\mathcal{F} = (\mathcal{D}, \mathcal{O})$ if:

$$\{unlock, refuse, free\} \cap \mathcal{O} \neq \emptyset$$

This suggests the link matrix of Fig. 5.

$\mathcal{L}(red) \widehat{=}$

unlock		x		x		x		x
refuse			x	x			x	x
free					x	x	x	x
pertinency	0	1	1	1	1	1	1	1

Figure 5. Link matrix of concept *Red*

In this representation the first three rows correspond to the possible configurations of operations of *red*. The 4th row measures the pertinency of each configuration considering concept *red*. We number the columns from 0 to 7.

From now, we can measure the pertinency of the data concept red, denoted by $\mathcal{L}_{\mathcal{F}}(red)$, in the formal context $\mathcal{F} = (\{..., red, ...\}, \{unlock, free\})$. As the pertinency of red was evaluated by an *or-combination* of its operations, it is then considered as pertinent for the formal context $\mathcal{F}$. In fact, the operations grouped by $\mathcal{F}$ correspond to the configuration of column 5 in $\mathcal{L}(red)$ where $\mathcal{L}_{\mathcal{F}}(red) = 1$.

Taking into account proper operations to a class concept d (Def. 3) leads to a link matrix $\mathcal{L}(d)$ defined by an *and-combination* of $\mathcal{O}_{\mathcal{P}}(d)$ because all proper operations must be associated with d, and an *or-combination* of the other operations ($\mathcal{I}[\{d\}] - \mathcal{O}_{\mathcal{P}}(d)$) because at least one operation must be associated with d. For example, $pass$ is proper to $PERSON$ and $unlock$ and $refuse$ are shared with data concept $DOOR$ (non-proper operations) the corresponding link matrix will be that of Fig. 6.

$\mathcal{L}(PERSON) \widehat{=}$

pass	$\perp$	x	$\perp$	x	$\perp$	x	$\perp$	x
unlock			x	x			x	x
refuse					x	x	x	x
Pertinency	0	1	0	1	0	1	0	1

Figure 6. Link matrix of concept *PERSON*

This means that the only contexts $\mathcal{F} = (\mathcal{D}, \mathcal{O})$ for which $PERSON$ is pertinent are the contexts where $pass \in \mathcal{O}$. It also means that the presence of the non-proper operations in the context does not influence its pertinency.

4.2. More precise metrics

We now propose a new estimation of a data concept pertinency which depends on the set of operations which will be grouped with a data concept in a context and also the priority given to the different kinds of accesses (a modification access may have a greater weight than a reading access, and a reading access may have a greater weight than a precondition access). Thus, the definition of the link matrix requires a more precise definition of the dependance of a data concept on its set of operations.

The link matrix is a usable form to encode our estimation for pertinency, especially when associating weights to kinds of operation accesses. A weight m is a measure given empirically to affect a priority to a specific kind of access. For example, considering a data concept d with $\mathcal{I}[\{d\}] = \{o_i, o_j\}$, and if m_i corresponds to a modification access of operation o_i and m_j to a read access of operation o_j to data concept d, then m_i should be greater than m_j and a context $\mathcal{F}_1 = (\{\ldots, d, \ldots\}, \{\ldots, o_i, \ldots\})$ will be preferred to a context $\mathcal{F}_2 = (\{\ldots, d, \ldots\}, \{\ldots, o_j, \ldots\})$ because $\mathcal{L}^w_{\mathcal{F}_1}(d) = \frac{m_i}{m_i+m_j} > \mathcal{L}^w_{\mathcal{F}_2}(d) = \frac{m_j}{m_i+m_j}$ ($\mathcal{L}^w_{\mathcal{F}}(d)$ is the pertinency evaluation of the data concept d in the context $\mathcal{F}$ using the weighted link matrix $\mathcal{L}^w(d)$). A context $\mathcal{F}_3 = (\{\ldots, d, \ldots\}, \{\ldots, o_i, \ldots, o_j, \ldots\})$ will be preferred to contexts $\mathcal{F}_1$ and $\mathcal{F}_2$ because the pertinency measure associated to d in $\mathcal{F}_3$ should be $\mathcal{L}^w_{\mathcal{F}_3}(d) = \frac{m_i+m_j}{m_i+m_j} = 1$.

For example, let w, r and p be the respective weights of a write access, a read access and a precondition access. The weights associated to operations $unlock, refuse$ and $free$ considering data concept red are then respectively p, w and w. Let $\mathcal{W} = p + w + w$. The weighted link matrix of red using an *or-combination* of $unlock, refuse$ and $free$ is given in Fig. 7.

For example, if $p = 0.20, r = 0.40$ and $w = 0.80$ then $\mathcal{W} = 1.80$ and $\mathcal{L}^w(red)$ is that of Fig. 8

This means that the best context for data concept red is a context grouping the three operations *unlock*, *refuse* and *free*. But a context $\mathcal{F}$ which groups only operations *refuse* and *free* may also be considered as suitable for data concept red because it gives a pertinency rate of about 90% ($\mathcal{L}^w_{\mathcal{F}}(red) \simeq 88.8\%$). Moreover, a context $\mathcal{F}'$ with only operation *unlock* must be avoided for data concept red because it gives a low pertinency rate ($\mathcal{L}^w_{\mathcal{F}'}(red) \simeq 11.1\%$).

In case of proper operations the weighted link matrix associates a binary value (0 or 1) to corresponding pertinency evaluation. For example, the weighted link matrix of data concept $PERSON$ is given in Fig. 9.

$\mathcal{L}^w(PERSON) \widehat{=}$

pass	$\perp$	p	$\perp$	p	$\perp$	p	$\perp$	p
unlock			p	p			p	p
refuse					p	p	p	p
Pertinency	0	$\frac{p}{\mathcal{W}}$	0	$\frac{p+p}{\mathcal{W}}$	0	$\frac{p+p}{\mathcal{W}}$	0	1

Figure 9. Weighted link matrix of data concept *PERSON* with $\mathcal{W} = p + p + p$

5. Context Model

At this point, we only considered pertinency with respect to a single data concept of a context. When several data concepts, associated to different contexts, are considered, adding an operation to one context will usually reduce the pertinency of another context. There is thus some trade-off to find in sharing operations and data concepts amongst classes.

A context model $\mathcal{G}$ is formed by a set of contexts and a list of link matrices corresponding to the considered data concepts in the model. To be translated into a class diagram a context model must satisfy the following properties:

- **Completeness**: All data concepts used by the B operations must take part in the contexts of $\mathcal{G}$
- **Operation cover**: All the operations of the B specifications are grouped in the contexts without redundancy.
- **Data concept pertinency**: All data concepts must be pertinent in each context $\mathcal{F} = (\mathcal{D}, \mathcal{O})$ of the context model:
  $$\forall\, d \cdot (d \in \mathcal{D} \Rightarrow \mathcal{L}_{\mathcal{F}}(d) = 1)$$
  When considering a weighted link matrix, $\mathcal{L}^w_{\mathcal{F}}(d)$ must be strictly positive.
- **Data concept Maximality**: There exists one maximal data concept in every context.

5.1. Derived context models

The first step to build a pertinent context model is to address the data concept maximality property which assesses that the context model is formed by two contexts containing the two maximal data concepts *DOOR* and *PERSON*. This allows to form two contexts which are: $\mathcal{F}_1 = (\{DOOR, \ldots\}, \{\ldots\})$ and $\mathcal{F}_2 = (\{PERSON, \ldots\}, \{\ldots\})$. In order to take into account the data concept pertinency property we start first by identifying proper operations of maximal data concepts and add them to the corresponding contexts. In fact, $\mathcal{O}_{\mathcal{P}}(PERSON) = \{pass\}$ and $\mathcal{O}_{\mathcal{P}}(DOOR) = \{lock,\ free\}$ which means that $\mathcal{F}_1 = (\{DOOR, \ldots\}, \{lock,\ free, \ldots\})$ and $\mathcal{F}_2 = (\{PERSON, \ldots\}, \{pass, \ldots\})$. Operations *unlock* and *refuse* can either be in $\mathcal{F}_1$ or in $\mathcal{F}_2$. They have a precondition access to data concepts *PERSON* and *DOOR*, and their link matrices give the same pertinency measure (equal to 1) when associated to $\mathcal{F}_1$ or $\mathcal{F}_2$. We notice that there exist 4 possibilities of their target context which leads to the following context models:

$$\begin{aligned}\mathcal{G}_1 :\ & \mathcal{F}_1 = (\{DOOR, \ldots\}, \{lock, free\})\\ & \mathcal{F}_2 = (\{PERSON, \ldots\}, \{pass,\ unlock,\ refuse\})\end{aligned}$$

$$\begin{aligned}\mathcal{G}_2 :\ & \mathcal{F}_1 = (\{DOOR, \ldots\}, \{lock, free,\ unlock,\ refuse\})\\ & \mathcal{F}_2 = (\{PERSON, \ldots\}, \{pass\})\end{aligned}$$

$$\begin{aligned}\mathcal{G}_3 :\ & \mathcal{F}_1 = (\{DOOR, \ldots\}, \{lock, free,\ unlock\})\\ & \mathcal{F}_2 = (\{PERSON, \ldots\}, \{pass,\ refuse\})\end{aligned}$$

$$\begin{aligned}\mathcal{G}_4 :\ & \mathcal{F}_1 = (\{DOOR, \ldots\}, \{lock, free,\ refuse\})\\ & \mathcal{F}_2 = (\{PERSON, \ldots\}, \{pass,\ unlock\})\end{aligned}$$

5.2. Context model pertinency

At this stage, the derived context models satisfy the data concept maximality and the operation cover properties and also the data concept pertinency for the data concepts added to the contexts. From now, we can dispatch the other data concepts and associate them to contexts $\mathcal{F}_1$ and $\mathcal{F}_2$ in order to reach the completeness property. This is done as much as possible using the weighted link matrices to find the best configurations and keep verified the data concept pertinency property. For example data concept *red* is accessed by the operations *unlock*, *refuse* and *free*. While operation *unlock* has a precondition access to data concept *red*, operations

$\mathcal{L}^w(red) \widehat{=}$	unlock		p		p		p		p
	refuse			w	w			w	w
	free					w	w	w	w
	Pertinency	0	$\frac{p}{\mathcal{W}}$	$\frac{w}{\mathcal{W}}$	$\frac{p+w}{\mathcal{W}}$	$\frac{w}{\mathcal{W}}$	$\frac{w+p}{\mathcal{W}}$	$\frac{w+w}{\mathcal{W}}$	1

Figure 7. Weighted link matrix of data concept *red* with $\mathcal{W} = p + w + w$

$\mathcal{L}^w(red) \widehat{=}$	unlock		0.20		0.20		0.20		0.20
	refuse			0.80	0.80			0.80	0.80
	free					0.80	0.80	0.80	0.80
	Pertinency	0	11.1	44.4	55.5	44.4	55.5	88.8	1

Figure 8. example of a weighted link matrix of data concept *red*

refuse and *free* modify it. Using the weighted link matrix of Fig. 8, the pertinency evaluations of data concept red with respect to the previously identified context models are:

$\mathcal{G}_1$:	$\mathcal{L}^w_{\mathcal{F}_1}(red) = 44.4\%$ $\mathcal{L}^w_{\mathcal{F}_2}(red) = 55.5\%$	$\mathcal{G}_2$:	$\mathcal{L}^w_{\mathcal{F}_1}(red) = 100\%$ $\mathcal{L}^w_{\mathcal{F}_2}(red) = 0$
$\mathcal{G}_3$:	$\mathcal{L}^w_{\mathcal{F}_1}(red) = 55.5\%$ $\mathcal{L}^w_{\mathcal{F}_2}(red) = 44.4\%$	$\mathcal{G}_4$:	$\mathcal{L}^w_{\mathcal{F}_1}(red) = 88.8\%$ $\mathcal{L}^w_{\mathcal{F}_2}(red) = 11.1\%$

The best place of data concept *red* is context $\mathcal{F}_1$ of the context model $\mathcal{G}_2$ because it groups the three operations *unlock*, *refuse* and *free* ($\mathcal{L}^w_{\mathcal{F}_1}(red) = 1$). Context $\mathcal{F}_1$ of model $\mathcal{G}_4$ is also suitable to data concept *red* because it associates two operations with a modification access to the data concept. This is not obvious when taking into account the other data concepts.

Table 1 gives the maximum pertinencies (column $max(\mathcal{G}_i)$) of all data concepts of the *AccessControl* specification when associated to context models $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ and $\mathcal{G}_4$.

Fig. 10 compares the four contexts and concludes that $\mathcal{G}_2$ is suitable to all data concepts. We notice that $\mathcal{G}_2$ is the less suitable configuration for data concept *sit*, but it can be acceptable because the pertinency of sit in $\mathcal{G}_2$ is still about 67%. Moreover, sit will not be encapsulated in a class because it is a relation between *PERSON* and *DOOR* and will be rather transformed into an association. The resulting class diagram of $\mathcal{G}_2$ is given in Fig. 3.

In order to give a ranking of these context models we compute the average pertinency measure $\mathcal{M}$ (the last row of table 1). $\mathcal{G}_2$ is the best context model because it has the greatest average pertinency measure ($\mathcal{M}(\mathcal{G}_2)$ = 92%). Although $\mathcal{M}(\mathcal{G}_4)$ is about 88%, pertinencies of data concepts red, $green$ and unl don't reach their maximum in $\mathcal{G}_4$. Thus, graphs such as that of Fig. 10 associated to metrics $\mathcal{M}$ are a useful information to identify the best configuration from those allowed by our tool.

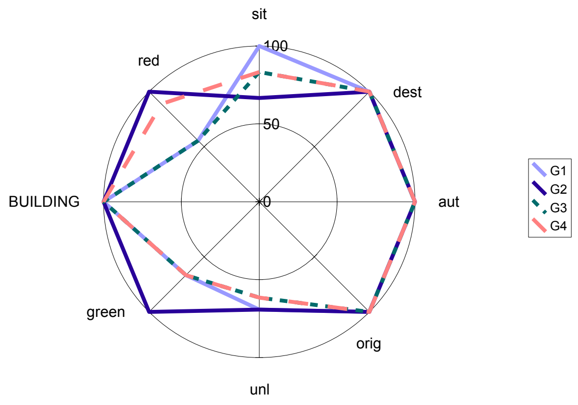


Figure 10. Comparison of $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$ and $\mathcal{G}_4$

Finally, from the 4 possible choices that the user may do from the "target" window of Fig. 2, the best one is to choose *PERSON* as a target of operation *pass* and associate *DOOR* to the other operations.

The derived context model is then formed by the two following contexts:
$\mathcal{F}_1$ = ({*DOOR*, *red*, *unl*, *green*}, {*lock*, *free*, *unlock*, *refuse*})
$\mathcal{F}_2$ = ({*PERSON*, *sit*, *orig*, *aut*, *dest*, *BUILDING*}, {*pass*})
and it can be further transformed into Fig. 3.

The transformation of contexts into classes and relations is discussed in [14]. Our tool then uses the *Dot* visualisation system [10] to layout the class diagram.

5.3. Further evaluation

This paper builds on principles of [12, 14] but introduces a new notion called link matrix in order to encode in a usable form our estimation of context pertinency. Each set of

Table 1. data concepts pertinency

	$\mathcal{G}_1$			$\mathcal{G}_2$			$\mathcal{G}_3$			$\mathcal{G}_4$		
	$\mathcal{F}_1$	$\mathcal{F}_2$	$max(\mathcal{G}_1)$	$\mathcal{F}_1$	$\mathcal{F}_2$	$max(\mathcal{G}_2)$	$\mathcal{F}_1$	$\mathcal{F}_2$	$max(\mathcal{G}_3)$	$\mathcal{F}_1$	$\mathcal{F}_2$	$max(\mathcal{G}_4)$
sit	0	100	100	33.3	66.6	66.6	16.6	83.3	83.3	16.6	83.3	83.3
red	44.4	55.5	55.5	100	0	100	55.5	44.4	55.5	88.8	11.1	88.8
BUILDING	0	100	100	0	100	100	0	100	100	0	100	100
green	33.3	66.6	66.6	100	0	100	66.6	33.3	66.6	66.6	33.3	66.6
unl	30.7	69.2	69.2	69.2	30.7	69.2	61.5	38.4	61.5	38.4	61.5	61.5
orig	0	100	100	0	100	100	0	100	100	0	100	100
aut	0	100	100	0	100	100	0	100	100	0	100	100
dest	0	100	100	0	100	100	0	100	100	0	100	100
Average ($\mathcal{M}$)			86.5%			92%			83.4%			87.6%

pertinent contexts form a context model which is then translated into a class diagram. For example, the class diagram issued from context model $\mathcal{G}_2$ is that of Fig. 3. We don't discuss in this paper how the formed contexts are translated into class diagrams, this is studied in [14], but we have proposed an approach to select the most pertinent set of contexts from those which are generated by our previous proposal [14].

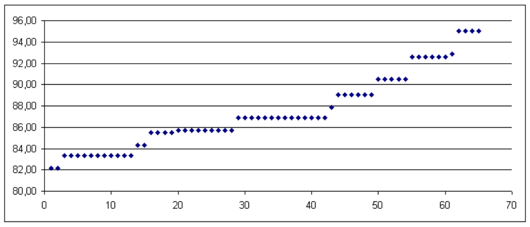
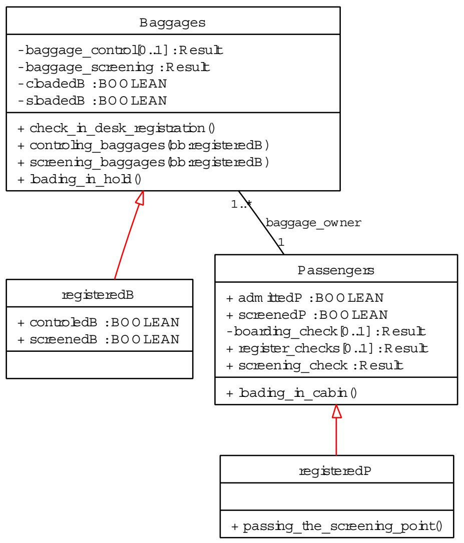


Figure 11. A ranking of the 65 context models issued from the *SecureFlight* specification

To evaluate our technique, we applied it to the *SecureFlight* specification. Our previous proposal leads to 65 possible configurations. This needs a lot of involvement from the user to select those which would be the more pertinent. Fig. 11 shows the computed pertinencies of the 65 possible configurations. As we can see 4 of these have an average pertinency measure equal to 95%. For illustration, Fig. 12 shows the class diagram issued from one of these best configurations. Thus, by using link matrices, we improve our wish to present a higher potential for automation. The only activity left to the user is to associate weights to reading, writing and precondition access and to choose amongst the candidates with the highest rank. In Fig. 8, we used a simple scale where modification access has twice the weight of read access, and read access has twice the weight of precondition access. Further experiments are needed to try other values and identify the most interesting weight values.

Using our technique, we were able to scale up from small B specifications (several dozens of lines) to medium size ones (several hundreds or thousand lines). Today, the largest B specification (i.e. the METEOR subway) is about 100,000 lines. Scaling up to such sizes may bring interesting new problems to our tool: if the number of configurations explodes, it may be impossible to evaluate the pertinency of each possible configuration. New techniques will then be needed to master this explosion of configurations.



Baggages
-baggage_control[0..1] :Result
-baggage_screening :Result
-cloadedB :BOOLEAN
-sloadedB :BOOLEAN
+ check_in_desk_registration()
+ controling_baggages(bb:registeredB)
+ screening_baggages(bb:registeredB)
+ loading_in_hold()
1..*
baggage_owner
1
registeredB
+ controledB :BOOLEAN
+ screenedB :BOOLEAN
Passengers
+ admittedP :BOOLEAN
+ screenedP :BOOLEAN
-boarding_check[0..1] :Result
+ register_checks[0..1] :Result
+ screening_check :Result
+ loading_in_cabin()
registeredP
+ passing_the_screening_point()

Figure 12. Class diagram issued from one of the best context models of the *SecureFlight* specification

6. Conclusion

Although formal methods ensure precision and consistency in software development, the actual practices of software engineering give a greater place to graphical approaches and object oriented notations because of their expression capabilities. This is due to the fact that understanding formal descriptions is often restricted to experts. In this paper we proposed an approach to overcome this limitation by providing complementary UML views from formal B specifications as an accompanying documentation. This documentation is first intended to be read by customers or certification authorities who are external to the formal development activities. It can also help formal method engineers to understand some structural properties of their specifications.

This paper improves the state of the art by exploiting a measure to evaluate the diagrams. This idea was applied to our approach, implemented by the B/UML tool, but it can also be adapted to similar approaches, such as [9], in order to interactively select the best rules to derive pertinent class diagrams. First, a link matrix is expressed to estimate the pertinency of data concepts with respect to their operations. Then, we select the groups of data concepts and operations which give the best pertinency rate. This approach was experimented with our tool in order to provide less involvement from the user to select the best diagram. An evaluation on the *SecureFlight* specification showed that the tool provides an effective help to face the large number of choices offered to the developer.

Still, further research is needed to help define weights in a more systematic way, and help evaluate whether a diagram is intuitive or not. Currently we are working on how our approach, initially defined to evaluate contexts, can be adapted to the evaluation of the generated class diagrams. We also believe that interesting improvements of our metrics can be inspired from works such as CK metrics [6] which evaluate object oriented design.

References

[1] J.-R. Abrial. *The B-book: assigning programs to meanings*. Cambridge Univ., 1996.

[2] J.-R. Abrial. System study: Method and example. www-lsr.imag.fr/B/Documents/ClearSy-CaseStudies/, 1999.

[3] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. METEOR: A successful application of B in a large project. In J. Wing, J. Woodcock, and J. Davies, editors, *Proceedings of FM'99: World Congress on Formal Methods*, number 1709 in LNCS, pages 369–387. Springer-Verlag, 1999.

[4] G. Bernhard and W. Rudolf. *Formal concept analysis*. Springer-Verlag, 1999.

[5] L. Casset. Development of an embedded verifier for java card byte code using formal methods. In *FME'02, Formal Methods Europe*, volume 2391 of *LNCS*. Springer-Verlag, 2002.

[6] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[7] M.-L. P. D. Bert and N. Stouls. GeneSyst: a Tool to Reason about Behavioral Aspects of B Event Specifications. Application to Security Properties. In *ZB 2005*, volume 3455 of *LNCS*, pages 299–318. Springer-Verlag, 2005.

[8] D. Essamé. La méthode B et l'ingénierie système. *TSI, (Technique et Science Informatiques)*, 23(7):929–938, 2004.

[9] H. Fekih, L. Jemni, and S. Merz. Transformation des spécifications B en des diagrammes UML. In *AFADL : Approches Formelles dans l'Assistance au Développement de Logiciels*, 2004.

[10] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, 30(11):1203–1233, 2000.

[11] A. Hammad, B. Tatibouet, J. Voisinet, and W. Weiping. From a B specification to UML statechart diagrams. In *4th Int. Conf. on Formal Engineering Methods (ICFEM'2002), LNCS 2495*, pages 511–522, China, 2002.

[12] A. Idani and Y. Ledru. Object Oriented Concepts Identification from Formal B Specifications. In *Proc. of 9th International Workshop on Formal Methods for Industrial Critical Systems*, volume 133 of *ENTCS*, pages 159–174. Elsevier, 2005.

[13] A. Idani and Y. Ledru. Dynamic Graphical UML Views from Formal B Specifications. *Information and Software Technology Journal*, 48(3), 2006. Elsevier.

[14] A. Idani, Y. Ledru, and D. Bert. Derivation of UML Class Diagrams as Static Views of Formal B Developments. In *Formal Methods and Software Engineering, 7th International Conference on Formal Engineering Methods, ICFEM 2005*, volume 3785 of *LNCS*, pages 37–51, Manchester, UK, November 2005. Springer-Verlag.

[15] R. Laleau and F. Polack. Coming and going from uml to b: A proposal to support traceability in rigorous is development. In *2nd International Conference of B and Z Users*, volume 2272 of *LNCS*, pages 517–534. Springer-Verlag, 2002.

[16] K. Lano, D. Clark, and K. Androutsopoulos. UML to B: Formal Verification of Object-Oriented Models. In *Integrated Formal Methods*, volume 2999 of *LNCS*, pages 187–206. Springer-Verlag, 2004.

[17] M. Leuschel and M. Butler. ProB: A Model Checker for B. In *FME 2003: Formal Methods*, volume 2805 of *LNCS*, pages 855–874. Springer-Verlag, 2003.

[18] G. Pouzancre. How to Diagnose a Modern Car with a Formal B Model? In *ZB 2003*, volume 2651 of *LNCS*, pages 98–100. Springer-Verlag, 2003.

[19] G. Pouzancre and J.-P. Pitzalis. Modélisation en B événementiel des fonctions mécaniques, électriques et informatiques d'un véhicule. *TSI, (Technique et Science Informatiques)*, 22(1):119–128, 2003.

[20] H. A. Sahraoui, W. L. Melo, H. Lounis, and F. Dumont. Applying concept formation methods to object identification in procedural code. In *IEEE Automated software engineering conference*, pages 210–218, 1997.

[21] C. Snook and M. Butler. UML-B: Formal modelling and design aided by UML. *ACM Transactions on Software Engineering and Methodology*. To appear.

[22] B. Tatibouet, A. Hammad, and J. Voisinet. From an abstract B specification to UML class diagrams. In *2nd IEEE International Symposium on Signal Processing and Information Technology (ISSPIT'2002)*, Marrakech, Morocco, Dec. 2002.

[23] T. A. Tilley, R. J. Cole, and P. W. Eklund. *A Survey of Formal Concept Analysis Support for Software Engineering Activities*. Springer-Verlag, 2005.