Refinement Patterns for an Incremental Construction of Class Diagrams

Boulbaba Ben Ammar* and Mohamed Tahar Bhiri+
Faculty of Sciences of Sfax, Sfax University, Sfax, Tunisia
*Boulbaba.Benammar@fss.rnu.tn
+Tahar_Bhiri@yahoo.fr

***Abstract*—Specifying complex systems is a difficult task, which cannot be done in one step. In the framework of formal methods, refinement is a key feature to incrementally develop more and more detailed models, preserving correctness in each step. Our objective is an incremental development, using the technique of refinement with proof for UML specifications. Indeed, UML suffers from two major weaknesses, namely, it is not based on a simple and rigorous mathematical foundation and it does not support the concept of refinement with proof of correction. To achieve this, we advocate a development framework combining the semi-formal features of UML/OCL and the formal one from B method. We chose the B formal language in order to benefit from existing work done on coupling between UML and B. In addition, we propose and formalize in B the refinement patterns that promote incremental development with proof of UML/OCL class diagrams. We illustrate our purpose by the description of some development steps of an access control system.**

Keywords-UML; OCL; refinement pattern; class diagram

I. INTRODUCTION

Refinement is a process to transform an abstract specification into a concrete one [17]. It aims to develop systems incrementally, which are correct by construction [18]. Refinement is defined in a rigorous way in various formal languages, such as B [18], Event-B [17], Communicating Sequential Processes (CSP) [8], Z and Object-Z [14]. The Unified Modeling Language (UML) [19] is an object-oriented modeling language widely used. It is a de-facto standard, allowing graphical visualization of models facilitating communication inter-actors. But, it does not support the concept of refinement. It has a dependency relationship stereotyped «refine» to connect a client (or refined concrete element) to a provider (abstract element). This relationship is subject to several interpretations [15] and does not provide methodological assistance related to how to refine existing UML models. In addition, UML does not allow the verification of a refinement relationship between two models. In a formal language, such as B, Event-B, CSP, Z and Object-Z, although the refinement relationship is well-defined and well-supported (generation of proof obligations, interactive prover, model checker and animator), an experienced designer finds more or less important difficulties in identifying the different levels of abstraction (an “optimal” refinement strategy) for carrying out the process of refinement. Solutions based on the concept of pattern --like the design patterns in Object-Oriented (OO) applications-- to guide the designer during the refinement process begin to appear covering both the horizontal refinement for application areas as reactive systems [17][26][27] and the vertical refinement under B. The horizontal refinement [17] consists in introducing new details in an existing model. Each introduction of details to a model leads to a new model, which must be coherent with the previous one. If it is the case, we said that the second model refines the first one. Horizontal refinement aims at the progressive acquisition, by successive refinement, of a coherent model from abstract specifications, leading to the abstract formal specification of future software or system.

The vertical refinement [17] consists in going from an abstract model to a more concrete one, for example by reducing the non-determinism. The concrete model is a realization of the abstract model. For example, the B Automatic Refinement Tool (BART) [1] tool associated with B offers refinement rules that can be used in the final stages of vertical refinement phase of a formal process development. B and Event-B do not distinguish between horizontal and vertical refinement. In fact, both refinements use two types of refinement allowed by B, i.e., data refinement and algorithms or control refinement [18].

In the following, we briefly present in Section 2 the existing approach for construction of class diagrams. The proposed approach is presented in Section 3. In Section 4, we present our catalog of refinement patterns used in an incremental specification development process. In Section 5, we illustrate the use of the proposed approach using an access control case study. Section 6 concludes this paper and proposes perspectives.

II. RELATED WORK

UML is a graphical modeling language reference, offering an important range of diagrams. Class diagram, which can express the static aspects of a system, are one of the most used diagrams. At the "heart" of the object modeling, it shows the classes and interfaces of a system and the various relationships between them. Approaches to construction and verification of class diagrams have been highlighted in several studies [4][7][10][13][20][21].

The decomposition unit of object-oriented systems is the concept of class. A coarse characterization of classes makes a distinction between the analysis classes belonging to the space of problems (external world in modeling course) and design classes and implementation belonging to the space of solutions.

Methodological works supporting the identification of the useful and relevant classes were completed. A method known as “Underline the names in the document of the requirements” is proposed in [13]. The results of the application of this method are very sensitive to the used

 ISBN: 978-1-61208-343-8

style. This can lead designers to omit useful classes while introducing classes, which are not justified.

Class, Responsibility, Collaboration (CRC) cards [20] are paper cards on which the designers evoke the potential classes according to their responsibilities and in the way in which they communicate. This technique promotes interaction within teams but its contribution to the identification of quality classes is uncertain.

Meyer [7] provides the general heuristics for classes discovery, based on the theory of Abstract Data Types (ADT). He defines different uses of inheritance justifying their uses.

The Business Object Notation (BON) method [21] introduces useful advices to identify the classes. It proposes two structural concepts (cluster and class), two strong conceptual relations (customer and inheritance) and a simple language of assertions expressing the semantic properties attached to the modeled elements.

Many analysts and designers use only the class diagrams. Others use development process allowing scheduling of several types of UML diagrams. Some development process with UML adopts an approach based on use case diagrams in order to draw class diagrams [28].

The design classes represent the architectural abstractions, which facilitate the production of elegant and extensible software structures. Design patterns, including those of Gang of Four (GoF) [10] promote the identification of these classes.

Semi-formal graphical notations (such as UML) are generally intuitive, but do not allow rigorous reasoning. On contrary, formal notations (such as B) provide mathematical proofs, but are not easy to understand. Several studies coupling between semi-formal and formal notations exist. Among these works, we studied profitably those linking UML to B [16]. Works of coupling between UML and B go to the combination of UML and B in a new language named UML-B [9].

By using the technique of refinement, the approach described by Ben Ammar et al. [4] allows of a UML/OCL class diagram showing all the formal properties of the future system. The obtained class diagram, containing the analysis classes, represents a coherent abstract model of the future system. Such a model can be concretized (identification of design and implementation classes) by applying the technique of refinement.

III. APPROACH TO DESIGN A SYSTEM IN A STEP-WISE MANNER

Our approach consists on the proposal of a catalogue of refinement patterns (see Section IV) for incremental development of UML class diagrams. These patterns are built to solve recurrent problems in the development of the static part of an OO application, such as: introduction of an intermediate class [3], reification of an attribute, an enrichment of an association, decomposition of an aggregate and the introduction of a new entity. These patterns are characterized by a precise framework composed of six parts showing the fundamental aspects of a refinement pattern. These parts are Intention, Motivation, Solution, Verification, Example and See also. In addition, the proposed refinement patterns are formalized into B specifications, using systematic rules of translation of UML into B [11][16]. This helps to identify precisely the conditions of applicability, the evolution of a UML class diagram and correctness of the refinement relationship. Such B formalization can be reused with advantage when instantiating these patterns by the designer. Thus, in a joint development UML/B, the designer selects and applies a refinement pattern on his abstract specification. Then, he obtains a new specification, which includes new properties related to the application of the refinement pattern. Verification of the correctness of the refinement relation between two specifications is entrusted to Atelier B tool [30].

In the following, we detail a new approach used for development of UML class diagrams guided by refinement patterns. Such development process allows establishing a UML class diagrams, which models the key concepts of the application and has properties considered to be coherent covering the constraints of the application resulting from its specifications. The process advocated has four steps: Rewriting of requirements, Refinement strategy, Abstract specification, and Refinement steps.

A. *Rewriting of requirements*

Currently, the specifications are often of poor quality. Abrial [27] criticize these specifications to be directed too towards a solution and to present mechanisms of realization to the detriment of the explicitness of the properties of the system to be conceived. We recommend to rewrite the specifications in order to put forward the properties of the future system and to facilitate the development of a suitable strategy of refinement. For that purpose, we use the recommendations of Abrial [17][27] for distinguishing the functional safety and liveness properties.

B. *Refinement strategy*

Rewriting of requirements facilitates the development of an adequate strategy of refinement. But, this does not guarantee obtaining an “optimal” strategy of refinement. Work making it possible to compare alternative strategies of refinement for a given scope of application, in case of the reactive systems, starts to appear [17][27].

C. *Abstract specification*

This stage aims to establish an abstract UML/OCL model described by a class diagram based on the refinement strategy previously defined. The UML/OCL class diagram product is translated into B in order to formally verify its coherence.

D. *Refinement steps*

The refinement process involves several steps. Each refinement step takes as inputs three parameters: the class diagram of level i, the proposed catalog of refinement patterns and the properties resulting from the specifications to be taken into account and produce as output the class diagram of level i+1 (see Figure 1).

 ISBN: 978-1-61208-343-8

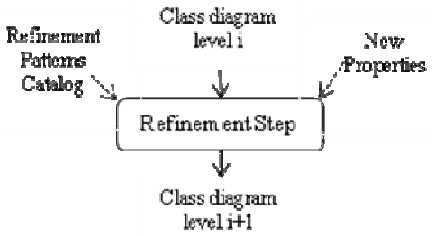


Figure 1. Step of refinement

The consideration of the properties, which guide the process of refinement, can be realized by applying refinement patterns. The formal verification of the correctness of the refinement step is entrusted to the AtelierB tool through the translation of two class diagrams in two B levels. The gluing invariant in B models making a link between these two levels (abstract and refined) can be established by reusing the B formalization of proposed refinement patterns. The refinement process terminates when all the explicit properties in the specification are taken into account in accordance with the adopted refinement strategy. Thus, ultimate UML/OCL class diagram obtained models the key concepts of the system to achieve. In addition, it contains the essential properties deemed formally consistent.

IV. REFINEMENT PATTERNS

Unlike architecture patterns [12], analysis patterns [22] and design patterns [10] a refinement pattern, has a dynamic character. Applied to a model of level i, a refinement pattern produces a model of level i+1. Recently, refinement patterns begin to appear for formalisms, such as Event-B [26], KAOS [2] and B [1]. In [3], we offer refinement patterns to solve recurring problems in incremental development of the static part of an OO application using UML/OCL. A refinement pattern has two parts: Specification (1) and Refinement (2). A specification describes the UML/OCL class diagram of level i. A refinement describes the UML/OCL class diagram of level i+1 produced by applying the corresponding refinement pattern on the model of level i. The proposed refinement patterns, presented later, are described in the same framework including six parts: Intention, Motivation, Solution, Verification, Example and See Also.

In the following, we detail the patterns only by the Intention, Motivation, Solution and See also.

A. Pattern 1: Class_Helper

1) Intention

It allows introducing a class Class_Helper between two classes considered important with respect to the refinement step considered. The direct relationship between the two major classes is refined by a path connecting these two classes through the intermediate class introduced.

2) Motivation

UML class diagram consists of four types of inter-class relationships: generalization (or inheritance), association, aggregation and dependence. In an incremental OO modeling, it is advantageous to start with abstract inter-class relationships. This subsequently facilitates the introduction of details via intermediate classes to refine these abstract relations.

3) Solution

(1)

(2)

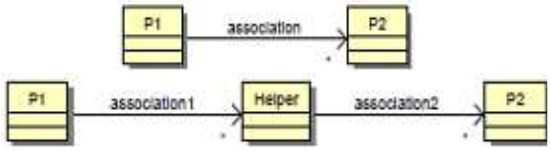


Figure 2. Class_Helper specification

4) See also

Class_Helper [3] the pattern depends on the nature of the relationship between two important classes P1 and P2: generalization, association, aggregation, composition and dependency. In addition, the intermediate class introduced Helper can be connected to P1 and P2 using the same kind of relationship or two relations of different nature. Class_Helper the pattern can be applied in reverse order of the concrete to the abstract. This process of abstraction - as opposed to refinement - can be profitably used in an activity of reverse engineering.

B. Pattern 2: Class_Attribute

1) Intention

When a class has an attribute modeling a concept considered interesting and with well-defined operations, this pattern allows to reify this attribute in a new class called Class_Attribute. An aggregation relationship is introduced between the enclosing class --aggregate-- and the class reifying the concerned attribute --component--.

2) Motivation

For reasons of simplification, at a high level of abstraction, a concept can be modeled as an attribute. Then, according to details from the specifications, the same concept can be retained as a class. This is justified by the identification of well-defined operations applicable on this concept by analyzing the introduced details. The type of the attribute is rather discrete: integer, enumerated or alphanumeric.

3) Solution

(1)

(2)

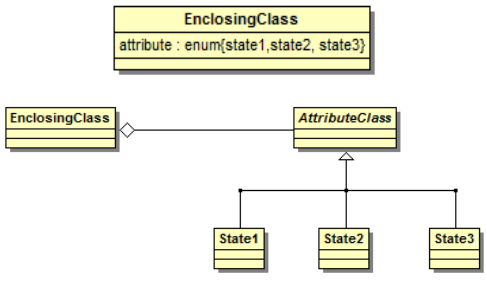


Figure 3. Class_Attribute specification

4) See also

The idea of reification of an attribute may be used with advantage in an activity of restructuring (or refactoring) of an existing OO models. Moreover, in [5], we proposed a

 ISBN: 978-1-61208-343-8

refactoring schema based on the reification of an attribute: introduction of the concept of delegation.

C. Pattern 3: Class_Decomposition

1) Intention

It allows detailing the responsibilities of an original class by introducing new classes. The original class and the resulting classes are connected by relations of generalization (inheritance). The number of the resulting classes is at least equal to one. This pattern favors a top-down modeling. The generalization covers mainly the following two cases [7]:

- Heritage subtype: You are an external model system in which a class of objects (external) can be decomposed into disjoint sub categories. We urge that the parent, A, be deferred so that it describes a set of objects not fully specified. The heir B can be effective or delayed.
- Restriction inheritance: Inheritance of restriction applies if the instances of B are among the instances of A, those that satisfy a constraint expressed in the invariant B and absent from the invariant A. A and B should both be deferred or both effective.

2) Motivation

In a top-down modeling approach, it is advantageous to start with a minimum number of classes. Sometimes we think that factoring operations can cause problems with implementation.

3) Solution

(1)

(2)

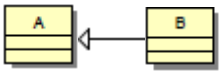


Figure 4. Class_Decomposition specification

4) See also

The pattern Class_Decomposition introduces the idea of a decomposition of a class via inheritance relationship. Both UML relationships: aggregation and composition can be used for the decomposition of an aggregate entity modeled by UML class.

D. Pattern 4: Class_NewEntity

1) Intention

It allows the introduction of UML class, which models a separate entity. The introduced class is related to other classes from abstract level through association relationships.

2) Motivation

In an incremental OO modeling, it is advantageous to start with a minimum number of entities called very abstract entities. This further promotes the introduction of details through less abstract or concrete entities, called (e.g., equipment) to go from the abstract world to the concrete world.

3) Solution

(1)

(2)

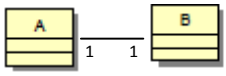


Figure 5. Class_NewEntity specification

4) See also

On the form, the two patterns Class_Helper and Class_NewEntity produce similar effects. But onthe content, they differ. In fact, they have two different gluing invariants. In addition, the pattern Class_NewEntity is oriented towards the horizontal refinement (specification stage) encouraging the construction step-by-step of a business model of the application, while the pattern Class_Helper is oriented towards the vertical refinement (design stage) promoting the gradual construction of a conceptual model of the application.

E. Pattern 5: Refinement_Operation

1) Intention

This pattern provides a passage from an abstract specification of operation into a more concrete one. It is inspired by formal development practices used in B method.

2) Motivation

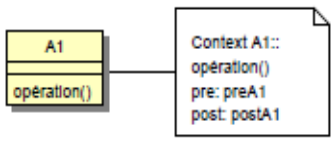
B method allows several types of refinement: data refinement, control refinement and algorithmic refinement. In control refinement, the following facts are observed:

- the operation to be refined retains the same signature,
- its precondition can be strengthened,
- the nondeterministic behavior, described by substitutions, can be reduced.

In UML framework, we can describe the control refinement using Object Constraint Language (OCL) notations for presenting both abstract and concrete specification of operation to be refined.

3) Solution

(1)



(2)

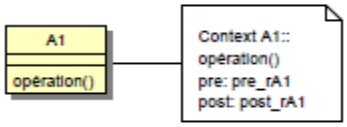


Figure 6. Refinement_operation specification

4) See also

The pattern Refinement_Operation introduced the idea of control refinement. In the same way, we can define a pattern of data refinement. This allows the introduction of concrete variables (data). In this case, a gluing invariant, which links abstract and concrete variables, should be explained. Both control and data refinement are not mutually exclusive; they can be operated in the same refinement step. It is obvious

 ISBN: 978-1-61208-343-8

that the pattern Refinement_Operation can be applied combining these two types of refinement.

F. Pattern 6: Class_Abstraction

1) Intention

The pattern Class_Abstraction introduced software qualities, such as efficiency, reusability and scalability in a software development guided by successive refinements. Thus, it allows factoring common properties - attributes, operations and relationships - of some classes within a founding class.

2) Motivation

In the development process, each entity is modeled by a class. But often, classes that are, in fact, variations of the same concept are encountered. Several classes of a class diagram have common characteristics. It is said that these independent classes can be derived from a common ancestor.

The idea is to improve the modeling, a better representation, facilitate data storage and thus avoiding redundant features. For that, we can factor these common features between the different classes into a new founding class.

3) Solution

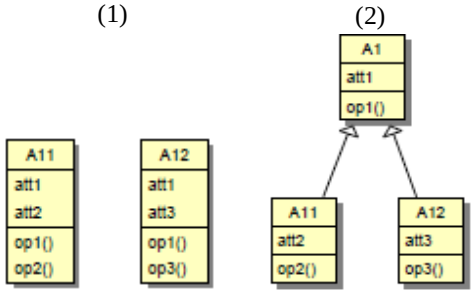


Figure 7. Class_Abstraction specification

4) See also

The pattern of change introduced by this pattern can be used with advantage in the process of refactoring to do to improve the structure (or quality) of an existing OO software. A refinement process with evidence rather favors obtaining a correct by construction software. The pattern Class_Abstraction advocates for the inclusion of other software qualities, such as efficiency, scalability from the initial phases of a development process guided by successive refinements. Besides, the risk of overlooking the efficiency quality in a process of refinement with proof is mentioned in [24].

G. Pattern 7: Class_Association

1) Intention

This pattern can increase the power of an association by considering both as an association relationship and an association class. This can be justified by the emergence of the specific details of the association relationship. Indeed, such details may be attached to extremities of the association. An association class can only exist if the association relationship exists.

2) Motivation

Sometimes, an association must own properties. These properties cannot be attached to the extremities of this association.

3) Solution

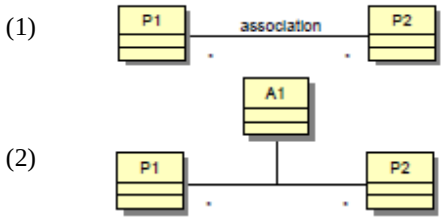


Figure 8. Class_Association specification

4) See also

The specification part of Class_Helper pattern is identical to the pattern Class_Association. However, their refinement parts are different.

V. EXAMPLE

Our objective is to develop a system to control the access of person to the various buildings of a workplace, inspired by [17]. In [17], this application is modeled in Event-B. In this work, we provide a joint development in UML / OCL and B of this application by using the proposed refinement patterns. Proof tools and animation associated with B are used to perform automated verification of UML /OCL graphical models. Control is carried out from the authorizations assigned to the concerned persons. An authorization allows a person, under the control of the system, to enter in some buildings and not in others. The authorizations are permanent, i.e., they cannot be modified during the operation of the system. When a person is inside a building, his exit must also be controlled so that it is possible to know, at any moment, who are in a given building. A person can move from a building to another only if these two buildings are interconnected. The communication between the buildings is done through one-way doors. Each door has an origin building and a destination building. A person may enter a building by crossing a door if it is unlocked. The doors being physically locked, a door unlocked for only one authorized person requiring entering the building. A green LED associated with each door is lit when the requested access is authorized, prerequisite for unlocking the door. Similarly, a red LED associated with each door is lit when the requested access is denied to the door. Each person has a magnetic card. Card readers are installed at each door to read the information on a card. Near each reader, there is a turnstile that is normally blocked; no one can cross it without the control of the system. Each turnstile is equipped with a clock, which determines in part its behavior.

A. Rewriting of requirements

Rewriting of requirements of the case study aims to highlight the properties of this application. In order to classify these requirements, we used the following labels:

- EQU-Equipment to reference the description of equipment used by the application.

 ISBN: 978-1-61208-343-8

- FUN-Equipment/Actor to reference an attached functionality of a device or an actor.
- MODEL-FUN-number to reference an assured by the application functionality.
- FUN-MODEL to reference the main function of the application.

In Table I, we will list the different requirements of the application of building access control. Each property is described by a relatively short text and a reference.

TABLE I. REWRITING OF REQUIREMENTS OF AN ACCESS CONTROL SYSTEM.

The system is responsible for controlling access of a number of people to several buildings.	FUN-MODELE
Each person is allowed to enter certain buildings (and not others). Buildings not recorded in this authorization are implicitly prohibited. This is a permanent assignment.	MODELE-FUN-1
Any person in a building is allowed to be there.	MODELE-FUN-2
The geometry of the building is used to define which buildings can communicate with each other and in what direction.	MODELE-FUN-3
A building does not communicate with itself.	MODELE-FUN-4
A person cannot move from a building where it is to a building where he wants to go if these two buildings communicate with each other.	MODELE-FUN-5
Any person authorized to be in a building should be allowed to go to another building that communicates with the first.	MODELE-FUN-6
The buildings are connected together by means of gates, which are one-way. We can therefore speak of origin and destination buildings for each door.	EQU-DOOR
A door cannot be taken if it is unlocked. A door can be unlocked for only one person at the same time. Conversely, any person involved in the unlocking of a door cannot be in one another.	FUN-DOOR-1
When a door is unlocked for a certain person, it is in the building behind the door in question. In addition, this person is allowed to go to the destination building of same door.	FUN-DOOR-2
When a door is unlocked for a certain person, it is in the building behind the door in question. In addition, this person is allowed to go to the destination building of same door.	FUN-PERSON
A green LED associated with each door.	EQU-GREENLIGHT
A green LED is lit when the requested access is allowed (pre requisite for unlocking the door).	FUN-GREENLIGHT
A red LED associated with each door	EQU-REDLIGHT
The red light of a door whose access has been denied.	FUN-REDLIGHT
The red and green lights of the same door cannot be turned on simultaneously.	FUN-LIGHT
Each person has a magnetic card that contains his permissions for different buildings.	EQU-CARD
Card readers are installed at each door to read the information on a card.	EQU-CARDREADER

B. Refinement strategy

Table II specifies the order of consideration of the properties and requirements of our case study: building access control. This defines our refinement strategy for incremental development of this application. The initial model is limited to the basic abstract properties of the application. Each refinement step includes a small number of properties from the abstract to the concrete. The refinement process ends when all properties from rewriting requirements were indeed taken into account. Equipment, such as door, card or LED that the application uses is introduced during the final stages of the adopted refinement process.

C. Abstract specification

We begin by developing a simple and very abstract class diagram that takes into account only the properties (FUN-MODELE, MODELE-FUN-2) (see Figure 9).

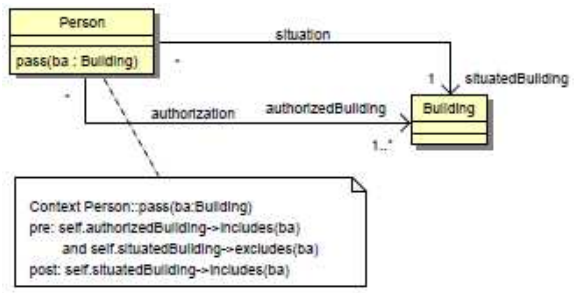


Figure 9. Initial class diagram

TABLE II. REFINEMENT STRATEGY

Model	Equipment and Function
Initial // First	FUN-MODELE, MODELE-FUN-2 // MODELE-FUN-1
Second	MODELE-FUN-3, MODELE-FUN-4, MODELE-FUN-5, MODELE-FUN-6
Third // Fourth	EQU-DOOR, FUN-DOOR-2 // FUN-DOOR-1, FUN-PERSON

 ISBN: 978-1-61208-343-8

Fifth	EQU-GREENLIGHT, FUN-GREENLIGHT, EQU-REDLIGHT, FUN-REDLIGHT
Sixth // Seventh	FUN-LIGHT // EQU-CARD, EQU-CARDREADER

D. Refinement steps

1) First refinement

In this step, we consider only the property (MODELE-FUN-1). This property indicates that the authorizations provided by the association "authorization" are permanent. For that, we applied data refinement based on pattern of Refinement_Operation. This refinement consists on the addition of a frozen constraint to the association "authorization" presented in Figure 9.

2) Second refinement

In this step, we inject into our system the properties (MODELE-FUN-3, MODELE-FUN-4, MODELE-FUN-5 and MODELE-FUN-6). These properties allow the introduction of the concept of communication between buildings. A person cannot move from one building to another only if the two buildings are interconnected.

The association “communication” is introduced into the class diagram as a recursive association on the class Building (see Figure 10). Such refinement requires a rewriting of the OCL expressions of the operation “pass”. Thus, we reused the refinement pattern Refinement_Operation.

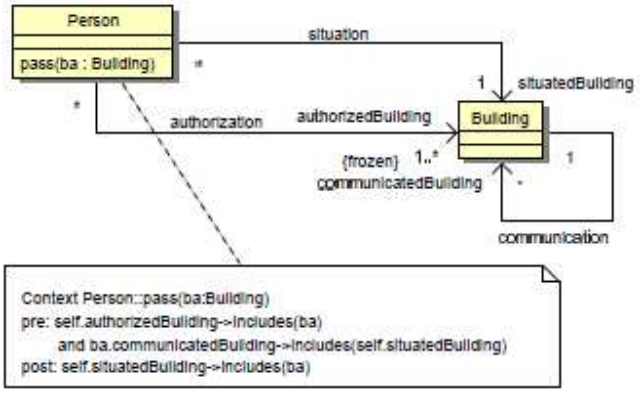


Figure 10. Second refinement

5) Third refinement

The third refinement consists in adding new equipment (EQU-DOOR and FUN-DOOR-2). A door can make the connection between two buildings. This leads us to define the concept of door: each door has original building and a destination building. Indeed, the communication between the buildings is through a door. Thus, we must remove the association "communication", introduced in the previous refinement, and replace it with two associations between origin Building and Door and between Door and destination Building. This change can be obtained by applying the refinement pattern Class_Helper with: communication as association; origin as association1; destination as association2; Door as Helper and Building as both P1 and P2. Finally, the property (FUN-DOOR-2) is that a door is a component of a building. Thus, we introduce a composition relationship between Door and Building (see Figure 11).

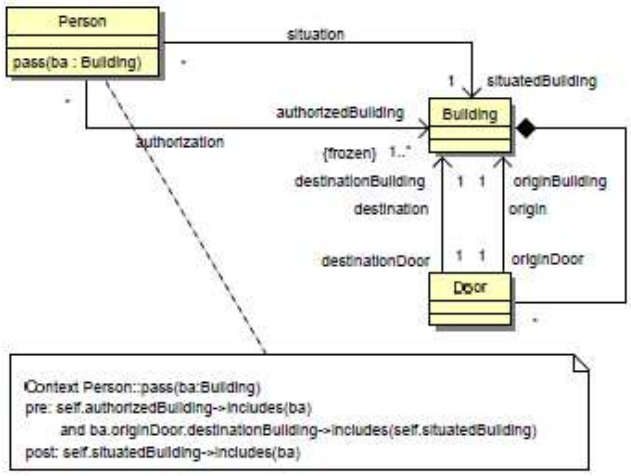


Figure 11. Third refinement

4) Fourth refinement

The fourth step of refinement consists on the definition of the functionality of the class Door introduced in the previous step (FUN-DOOR-1).

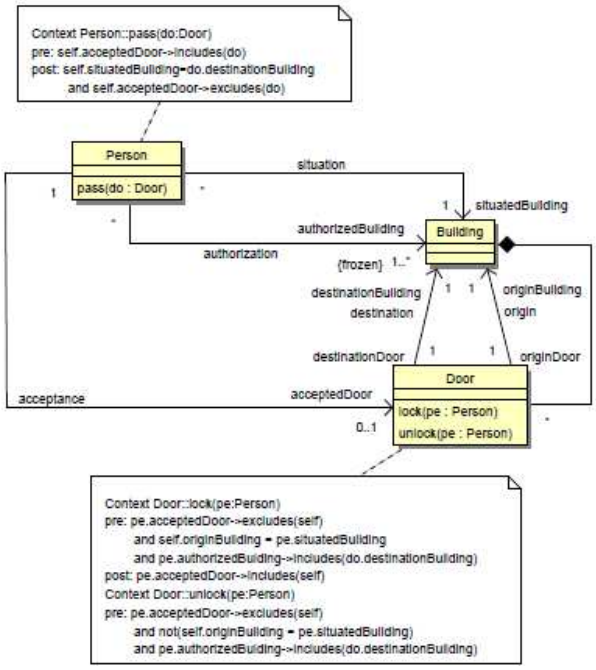


Figure 12. Fourth refinement

Such a transformation requires the revision of the semantics of the operation "pass". Indeed, property (FUN-

PERSON) is that a person must appear before a door to move from one building to another. This requires the introduction of an association "acceptance" between Person and Door. Thus, the operation "pass" should not take an instance of the class Building as formal parameter but rather an instance of the class Door. For this, we apply the refinement pattern Refinement_Operation to generate the class diagram presented in Figure 12.

5) Fifth refinement

In this step, we consider the properties (EQU-GREENLIGHT, FUN-GREENLIGHT, EQU-REDLIGHT and FUN-REDLIGHT). These properties define two new classes with their characteristics as components of the class Door. The application of refinement pattern Class_Decomposition with composition as a relationship between Door and RedLight and GreenLight as components generates the class diagram shown in Figure 13.

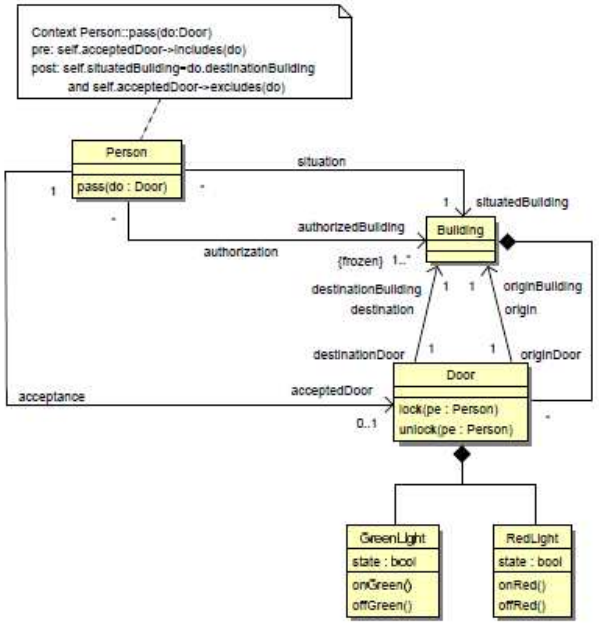


Figure 13. Fifth refinement

6) Sixth refinement

In this step, we note the similarity between the two classes RedLight and GreenLight (FUN-LIGHT). Thus, we decided to factor the common properties between these two classes. The application of refinement pattern Class_Abstraction generates the class diagram shown in Figure 14. The pattern Class_Abstraction allows introducing a new class named Light, which groups common properties between GreenLight and RedLight.

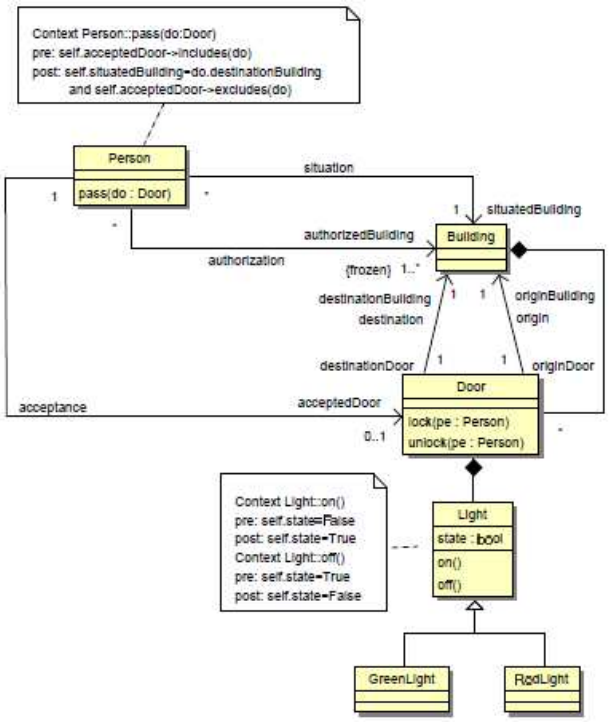


Figure 14. Sixth refinement

7) Seventh refinement

The last properties (EQU-CARD and EQU-CARDREADER) will be taken into account in this final stage of refinement.

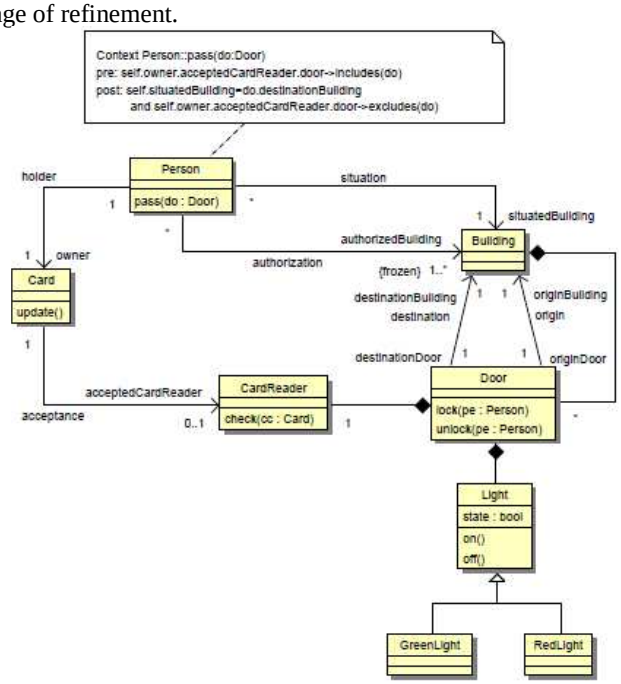


Figure 15. Seventh refinement

Two intermediate classes can be introduced:

- the class Card associated with each person,

- the class CardReader associated with each door of a building.

These classes are related as follows: Card is connected to Person, Card is connected to CardReader and CardReader is connected to Door.

The application of refinement pattern Class_Helper on the association "acceptance" generates the class diagram of Figure 15.

E. Verification of the obtained system

Formal verification of such refinements can be exploited if the language is equipped with formal refinement machinery, allowing the proof of the correctness of the refined specification relative to the abstract one. We proposed to use B for this purpose, using systematic derivation rules from UML into B. Such a translation of UML into B uses profitably the B formalization of the proposed refinement [6]. Indeed, the properties described in the form of B invariant --including gluing invariant-- are retrieved and instantiated when translating UML into B. As an illustration, Figure 16 and 17 shows the B formalization of pattern Class_Helper introduced in Section 4.

MACHINE
B_Class_Helper_a
SETS
OBJECTS = {p11, p12, p13,
p21, p22, p23, h1, h2, h3}
ABSTRACT_CONSTANTS
P1, P2
PROPERTIES
P1 ⊆ OBJECTS & P2 ⊆ OBJECTS ∧
P1 ∩ P2 = Ø ∧ P1 = {p11, p12, p13} ∧
P2 = {p21, p22, p23}
VARIABLES
p1, p2, association
INVARIANT
p1 ⊆ P1 & p2 ⊆ P2 ∧
association ∈ p1 ↔p2
INITIALISATION
p1 := {p11,p12, p13} ||
p2 := {p21, p22, p23} ||
association := {p11↦p21, p11↦p22,
p11↦p23, p12↦p21, p12↦p22,
p12↦p23, p13↦p21, p13↦p22,
p13↦p23}
END

Figure 16. B formalization of pattern Class_Helper

REFINEMENT
B_Class_Helper_r
REFINES
B_Class_Helper_a
ABSTRACT_CONSTANTS
HELPER
PROPERTIES
HELPER ⊆ OBJECTS ∧ HELPER ∩ P1 = Ø∧
HELPER ∩ P2 = Ø∧ HELPER = {h1, h2, h3}
ABSTRACT_VARIABLES
p1, p2, helper,
association1, association2
INVARIANT
helper ⊆ HELPER ∧
association1 ∈ p1↔helper ∧
association2 ∈ helper↔p2 ∧
ran(association1) = dom(association2) ∧
/∗Gluing Invariant∗/
dom(association) = dom(association1) ∧
ran(association) = ran(association2) ∧
ran(association1) = dom(association2) ∧
association = (association1;association2)
INITIALISATION
p1:={p11,p12,p13} k p2:={p21,p22,p23} ||
helper := {h1, h2, h3} ||
association1:={p117↦h1,p117↦h2,p117↦h3,
p127↦h1, p127↦h2, p127↦h3,
p137↦h1, p137↦h2, p137↦h3} ||
association2:={h17↦p21,h17↦p22,h17↦p23,
h27↦p21, h27↦p22, h27↦p23,
h37↦p21, h37↦p22, h37↦p23}
END

Figure 17. B formalization of pattern Class_Helper

The abstract machine B_Class_Helper_a formalizes the abstract level of Class_Helper pattern using the systematic translation rules of UML to B [11], while B_Class_Helper_r machine formalizes the refined level of the same pattern. The link between these two levels is described by the REFINES clause. Gluing invariant introduced in B_Class_Helper_r machine guarantees the correction of the refinement relation between the two levels of Class_Helper pattern. Formal verifications on the B models corresponding to UML/OCL class diagram are related to the coherence of the initial abstract model and the correction of each refinement step. They call the generator of proof obligations (conjectures to prove) and provers in B platform. The correction of B models, respecting requirements, is forward to the ProB tool [29], allowing animation and model checking.

TABLE III. TABLE OF THE STATE OF B SPECIFICATIONS

	nPO[1]	nPRi[2]	nPRa[3]	uUn[4]	%Pr
Initial model	9	1	8	0	100
Second refinement	4	0	4	0	100
Third refinement	7	0	7	0	100
Fourth refinement	12	3	9	0	100
Fifth refinement	12	2	10	0	100
Seventh refinement	26	0	26	0	100

[1] Number of Proof Obligations
[2] Number of Proof Obligations proved Interactively
[3] Number of Proof Obligations proved Automatically
[4] Number of Proof Obligations Unproved

Table III summarizes the proof obligations associated with our case study. The seven proposed refinement promote essentially the development of class diagrams correct by construction. However, the designer could improve the structure, without changing the semantic aspects of the class diagram obtained by refinement using wisely the refactoring technique [23][25].

VI. CONCLUSION AND FUTURE WORK

The main idea of this work is to propose intuitive refinements as patterns, providing a basis for tools supporting the refinement-driven modeling process. In this paper, we have presented our catalogue of refinement patterns. Formal verification of such refinements can be exploited if the language is equipped with formal refinement machinery, allowing the proof of the correctness of the refined specification relative to the abstract one. We proposed to use B for this purpose, using systematic derivation rules from UML into B. The proposed refinement patterns promote the identification of analysis classes that model the key concepts, resulting from requirements.

Currently, we are exploring the following two tracks: proposal of refinement patterns oriented design by retrieving and adapting ideas from GoF patterns [10]; proposal of refinement patterns oriented implementation, using the object-oriented modeling universal data structures (Eiffel) [7]. The next step of this work consists of automating detecting and application of patterns in an appropriate framework. In addition, we proposed a new approach, allowing finding a refinement strategy for the development of UML class diagrams guided by the refinement patterns. An interesting idea is to preserve the history of pattern application in development case studies in order to have a traceability of the development process, allowing to back-track on previous decisions.

REFERENCES

[1] A. Requet, “BART: A Tool for Automatic Refinement,” ABZ, London, UK, September, 2008, pp. 345-345.

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

[3] B. Ben Ammar, M.T. Bhiri, and A. Benhamadou, “Refinement Pattern: Introduction of intermediate class Class_Helper,” Conférence en IngénieriE du Logiciel, CIEL, Rennes, France, Jun, 2012, pp. 1-6.

[4] B. Ben Ammar, M.T. Bhiri, and J. Souquières, “Event modeling for construction of class diagrams,” RSTI - ISI, vol. 13, no. 3, 2008, pp. 131–155.

[5] B. Ben Ammar, M.T. Bhiri, and J. Souquières, “Refactoring pattern of class diagrams based on the notion of delegation,” 7éme atelier sur l'Evolution, Réutilisation et Traçabilité des Systèmes d’Information, ERTSI, couplé avec le XXVI éme congrès INFORSID, Fontainebleau, France, May, 2008, pp. 1-12.

[6] B. Ben Ammar, Contribution to the Systems Engineering: Refinement and Refactoring of UML specifications, Editions universitaires europeennes, 2012.

[7] B. Meyer, Object-oriented software construction, Prentice Hall, 1997.

[8] C. A. R. Hoare, “Communicating sequential processes,” Communications of the ACM, vol. 21, no. 8, 1978, p. 666-677.

[9] C. Snook and M. Butler. 2006, “UML-B: Formal modeling and design aided by UML,” ACM Trans. Softw. Eng. Methodol. vol. 15, no. 1, January, 2006, pp. 92-122.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns, Addison-Wesley, 1995.

[11] E. Meyer and J. Souquières, “A Systematic Approach to Transform OMT Diagrams to a B Specification,” Proceedings of the World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September,1999, pp. 875-895.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: a system of patterns, John Wiley and Sons, 1996.

[13] G. Booch, “Object-Oriented Development,” IEEE Trans. Software Eng., vol. 12, no. 2, 1986, pp. 211–221.

[14] G. Smith, The Object-Z Specification Language. Kluwer Academic Publishers, 2000.

[15] H. Habrias and C. Stoquer, “A formal semantics for UML refining,” XII Colloque National de la Recherche en IUT, CNRIUT'06, Brest, France, Jun, 2006.

[16] H. Ledang, “Automatic Translation from UML Specifications to B,” Proceedings of the 16th IEEE international conference on Automated software engineering, San Diego, USA, November, 2004, pp. 436-440.

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

[18] J. R. Abrial, The B Book - Assigning Programs to Meanings, Cambridge University Press, 1996.

[19] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual. 2 Boston, MA: Addison-Wesley, 2005.

[20] K. Beck and W. Cunningham, “A laboratory for teaching object-oriented thinking,” ACM SIGPLAN Not., vol. 24, no. 10, 1989, pp. 1–6.

[21] K. Waldèn and J. M. Nerson, Seamless object-oriented software architecture: analysis and design of reliable systems, Prentice-Hall, Inc., 1995.

[22] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley Professional, 1996.

[23] M. Fowler, Refactoring: Improving the Design of Existing Code.Boston, MA, USA: Addison-Wesley, 1999.

[24] M. Guyomard, “Specification and refinement using B: two pedagogical examples,” ZB2002 4th International B Conference, Education Session Proceedings, Grenoble, France, January, 2002.

[25] R. Straeten, V. Jonckers, and T. Mens, “A formal approach to model refactoring and model refinement,” Software and System Modeling (2), 2007, pp. 139-162.

[26] T. S. Hoang, A. Furst, and J. R. Abrial, “Event-B Patterns and Their Tool Support,” Software Engineering and Formal Methods, International Conference on, Hanoi, Vietnam, November, 2009, pp. 210-219.

[27] W. Su, J. R. Abrial, R. Huang, and H. Zhu, “From Requirements to Development: Methodology and Example,” The 13th International Conference on Formal Engineering Methods, ICFEM, Durham, United Kingdom, October, 2011, pp. 437-455.

[28] X. Castellani,“Cards stages of study of UML diagrams, Payment orders of these studies,” Technique et Science Informatiques, vol. 21, no. 8, 2002, pp. 1051–1072.

[29] The ProB Animator and Model Checker, User Manual, http://www.stups.uni-duesseldorf.de/ProB/index.php5/User_Manual, 2013.

[30] Clearsy System Engineering, Atelier B, User Manual, Version 4.0, http://www.tools.clearsy.com/resources/User_uk.pdf, 2010.

 ISBN: 978-1-61208-343-8