

On the Definition of UML Refinement Patterns

Claudia Pons

LIFIA – Computer Science Faculty – UNLP
La Plata, Buenos Aires, Argentina
cpons@info.unlp.edu.ar

Abstract. In this article we describe an approach towards increasing the robustness of the UML refinement machinery. The aim of this work is not to formalize the UML notation itself, but to substantiate a number of intuitions about the nature of possible refinement relations in UML, and even to discover particular refinement structures that designers do not perceive as refinements in UML.

1. Introduction

The stepwise refinement technique facilitates the understanding of complex systems by dealing with the major issues before getting involved in the details. It consists of developing the system through different levels of abstraction. The system under development is first described by a specification at a very high level of abstraction. A series of iterative refinements may then be performed with the aim of producing a specification, consistent with the initial one, in which the behavior is fully specified and all appropriate design decisions have been made.

Stepwise software development can be fully exploited only if the language used to create the specifications is equipped with formal refinement machinery, making it possible to prove that a given specification S_C is a refinement of another specification S_A , or even to calculate possible refinements from a given specification. This refinement machinery is present in most formal specification languages such as Z [6], B [9], and the refinement calculus [2]. Besides, some restricted forms of programming languages can also be formally refined [4]. But, in the standard specification language UML [13], the refinement machinery has not reach a mature state yet.

Consequently, we believe that any effort made towards increasing the robustness of the UML refinement machinery is a valuable task. To reach this goal most research on the formalization of UML refinements adhere to the approach of mapping the graphical notation into a formal domain where properties are defined and analyzed. For example the works presented in [5], [7], [1], [19], [10], [11] among others. Those “*informal-to-formal*” approaches are appropriate to discover and correct inconsistencies and ambiguities of the graphical language, and in most cases they allow us to verify and calculate refinements of (a restricted form of) UML models. However, such approaches are non-constructive: they provide no feedback in terms of UML and consequently, they provide little insight on a UML refinement methodology. Besides, such approaches are unable to put in evidence some weaknesses of the graphical language, such as the lack of notation and the presence of hidden concepts [15][16].

Thus, we decided to explore an alternative approach, called “*formal-to-informal*”, as a complement to the former. According to this approach a formally defined refinement methodology is immersed into a UML-based development. Concretely, well founded refinement structures in

the Object-Z formal language provide inspiration to define refinement structures in the UML, which are (intuitively) equivalent to their respective inspiration sources.

The structure of this document is as follows: first, sections 2 serves as a brief introduction to the issue of refinement specification in Object-Z and UML 2.0. Section 3 presents our proposal, describing a catalog of UML refinement patterns. Finally, section 4 presents a description of related work and conclusions.

2. Refinements Specification and Verification in Object-Z and UML

In Object-Z [18], a class is represented as a named box with zero or more generic parameters. The class schema may include local type or constant definitions, at most one state schema and an initial state schema together with zero or more operation schemas. These operations define the behavior of the class by specifying any input and output together with a description of how the state variables change. Refinement is formally addressed in the context of Object-Z specifications [6] as follows:

Definition (Downward Simulation in Object-Z): an Object-Z class C is a refinement (through downward simulation) of the class A if there is a *retrieve relation* R on $A.State \wedge C.State$ so that every visible abstract operation Aop is recasted into a visible concrete operation Cop thus the following holds:

- (*Initialization*) $\forall C.State \bullet C.init \Rightarrow (\exists A.State \bullet A.init \wedge R)$
- (*Applicability*) $\forall A.State; C.State \bullet R \Rightarrow (preAop \Rightarrow preCop)$
- (*Correctness*) $\forall A.State; C.State; C.State' \bullet R \wedge preAop \wedge Cop \Rightarrow \exists A.State' \bullet R' \wedge Aop$

This definition allows preconditions to be weakened and non-determinism to be reduced.

On the other hand, the standard modeling language UML [13] provides an artifact named *Abstraction* (a kind of Dependency) with the stereotype <<refine>> to explicitly specify the refinement relationship between UML named model elements. In the UML metamodel an Abstraction is a directed relation from a *client* (or clients) to a *supplier* (or suppliers) stating that the client (the refinement) depends on the supplier (the abstraction). The Abstraction artifact has a meta-attribute called *mapping* designated to record the abstraction/implementation mappings (i.e., the counterpart to the Object-Z *retrieve relation*), which is an explicit documentation of how the properties of an abstract element are mapped to its refined versions, and on the opposite direction, how concrete elements can be simplified to fit an abstract definition. The mapping contains an expression stated in a given language.

3. Refinement Patterns Catalog

In this section we describe a catalog of well-founded Object-Z refinement patterns, each of them giving origin to a list of several UML refinement patterns. The proliferation of patterns on the UML side is due to the fact that UML is a multi-perspective language that makes it possible to specify a system from more than one point of view, offering the possibility of creating a number of overlapped specifications, where each specification is focused on a single perspective (e.g., structure, collaboration, interaction, etc.). This means that each single Object-Z refine-

ment pattern can be analyzed from a number of perspectives, which give rise to a number of UML refinement structures, one for each perspective.

In the catalog, the description of each pattern consists of five sections: *DESCRIPTION* (a general and language-independent definition of the pattern is presented); *EXAMPLE* (an Object-Z instantiation of the pattern is specified); *FORMALIZATION* (the pattern is mathematically specified in Object-Z); *UML REALIZATION* (a list of UML instantiations of the pattern is presented) and *DISCUSSION* (problems encountered to realize the pattern are described and analyzed). For space limitation we present only a single entry of the catalog:

The State Refinement Pattern:

DESCRIPTION: a State Refinement takes place when the data structures which were used to represent the objects in the abstract specification are replaced by more concrete or suitable structures; operations are accordingly redefined to preserve the behavior .

EXAMPLE : in a flight booking system (figure 1), each flight is abstractly described by the quantity of free seats in its cabin; then a refinement is produced by recording the total capacity of the flight together with the quantity of reserved seats. In both specifications a Boolean attribute is used to represent the state of the flight (open or canceled). The available operations are `reserve` to make a reservation of one seat and `cancel` to cancel the entire flight. The retrieve relation *R* establishes the connection between both specifications.

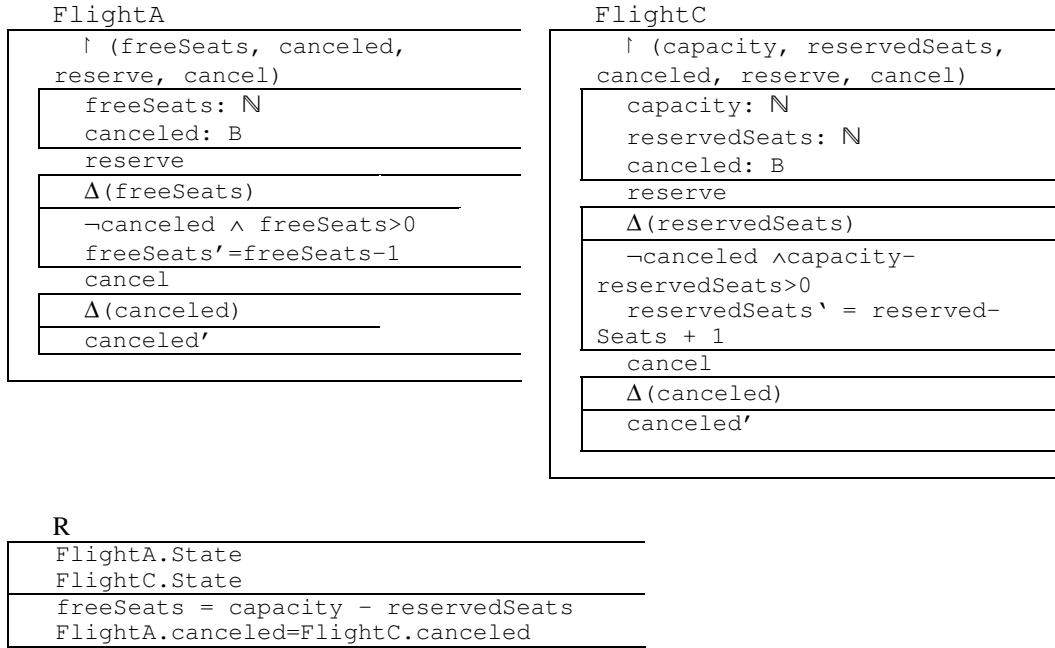


Figure 1. State refinement in Object-Z classes

FORMALIZATION: by applying the definition of downward simulation in Object-Z [6], presented in section 2, it is possible to verify the refinement conditions for the initialization and for every operation via the retrieve relation *R*:

Initialization

$\forall \text{FlightC.State} \bullet \text{FlightC.init} \Rightarrow (\exists \text{FlightA.State} \bullet \text{FlightA.init} \wedge R)$

Applicability (only a part is showed)

$\forall \text{FlightA.State}; \text{FlightC.State} \bullet R \Rightarrow (\text{pre reserveA} \Rightarrow \text{pre reserveC})$

Correctness (only a part is showed)

$\forall \text{FlightA.State}; \text{FlightC.State}; \text{FlightC.State}' \bullet R \wedge \text{pre reserveA} \wedge \text{reserveC}$
 $\Rightarrow \exists \text{FlightA.State}' \bullet R' \wedge \text{reserveA}$

UML REALIZATIONS:

In this section we describe a single UML instantiations of the State Refinement Pattern: State Refinement in Class Diagrams. Other instantiations of this pattern can be observed in Use Cases and other UML Classifiers with attributes and operations, but they are not included in the paper for space limitations.

The UML specification of the state refinement of the class Flight is depicted in figure 2; a refinement relationship connects the abstract to the concrete specifications. The OCL language [14] [17] has been used to specify the operation's pre and post conditions. The mapping attached to the abstraction relationship is expressed in an OCL-like language (a discussion on the mapping's language issue is included below).

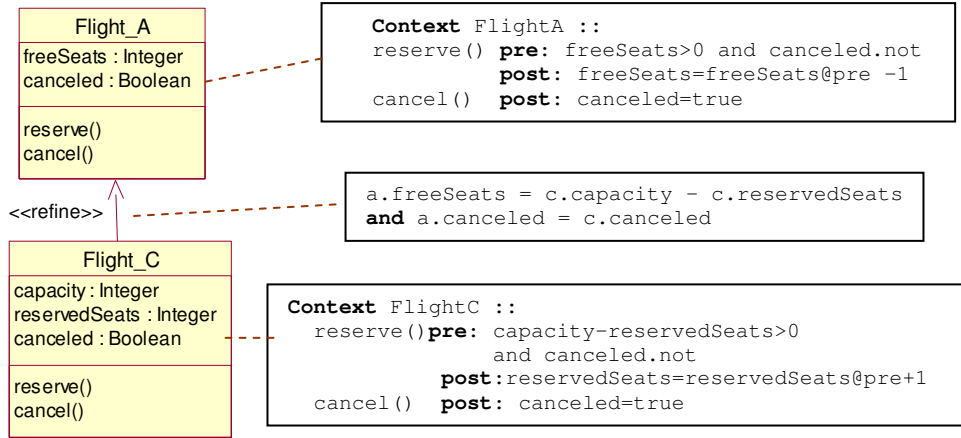


Figure 2. State refinement in UML class diagram

DISCUSSION:

Issues on the syntax to specify the retrieve relation: Graphically, the abstraction mapping describing the relation between the attributes in the abstract element and the attributes in the concrete element is attached to the refinement relationship; however, OCL expressions can only be written in the context of a Classifier, but not of a Relationship. Then, if we want to use the OCL to express the abstraction mapping we need to determine which the context of the expression is. On the Z side, the context of the abstraction mapping is the combination of the abstract and the concrete states (i.e., $A.State \wedge C.State$); however, a combination of Classifiers is not an OCL legal context; consequently we might write the mapping in the context of the abstract (or the concrete) classifier only, in the following way:

```
Context a:FlightA def:mapping(c : FlightC) : Boolean =
a.freeSeats = c.capacity -c.reservedSeats and a.canceled = c.canceled
```

The transformation from the pseudo-OCL expressions in figures 2 to their corresponding legal OCL expressions above can be generically defined in the following way: let *d* be a refine relationship with meta-attributes *d.supplier* (the abstract classifier), *d.client* (the concrete classifier) and *d.mapping* (the pseudo OCL expression specifying the mapping). We derive a Boolean operation definition in the context of the abstract classifier:

```
Context a:anAbstractElement def:mapping(c:aConcreteElement):Boolean =
aBoolOclExp
```

Where *anAbstractElement*, *aConcreteElement* and *aBoolOclExp* are replaced by *d.supplier.name*, *d.client.name* and *d.mapping.body*, respectively.

Issues on the verification process: Verification heuristics can be defined for this refinement pattern. On the one hand, to verify the refinement conditions we can translate the UML diagram back to Object-Z using already developed strategies such as the one proposed by Kim and Carington in [8]. Then, verification is carried out on the formal specification. Alternatively, we might remain on the UML+OCL side by defining refinement conditions in OCL in a similar style to the Object-Z refinement conditions. For example, we translate the following Object-Z applicability condition,

```
 $\forall \text{FlightA.State; FlightC.State} \cdot R \Rightarrow (\text{pre reserveA} \Rightarrow \text{pre reserveC})$ 
```

to an (intuitively) equivalent OCL expression:

```
TupleTypeA.allInstances->forAll(a|TupleTypeC.allInstances-> forAll(c|
  a.mapping(c) implies (a.freeSeats>0 and not a.canceled)
  implies(c.capacity-c.reservedSeats>0 and not c.canceled)))
```

This resulting expression can be evaluated using a regular OCL evaluator, providing positive evidence on the *applicability* of the refinement. The same procedure can be carried out to prove the *correctness* of the refinement. Translation steps can be fully automated in all situations where the object state consists of primitive values, which can be transformed to an OCL Tuple-Type.

4. Related Work and Conclusion

Boiten and Bujorianu in [3] explore refinement indirectly through unification; the formalization is used to discover and describe intuitive properties on the UML refinements. On the other hand, Liu, Jifeng, Li and Chen in [12] use a formal specification language to formalize and combine UML models. Then, they define a set of refinement laws of UML models to capture the essential nature, principles and patterns of object-oriented design, which are consistent with the refinement definition. The strategy we propose in this article apart from providing formal evidence on the presence of refinement structures in object-oriented designs made it possible to reveal hidden refinements and to discover weaknesses of the UML language that prevent designers from specifying frequently occurring forms of refinement. Besides, the understanding of refinement patterns is more precise, since each pattern is described from both an intuitive and a mathematical point of view. Finally, the overall contribution of this research is to clarify the abstraction/refinement relationship in UML models, providing basis for tools supporting the refinement driven modeling process.

5. References

- [1] Astesiano E., Reggio G. An Algebraic Proposal for Handling UML Consistency”, Workshop on Consistency Problems in UML-based Software Development. UML Conference (2003).
- [2] Back, R. & von Wright, J. *Refinement calculus: a systematic introduction*, Graduate texts in computer science, Springer Verlag. (1998)
- [3] Boiten E.A. and Bujorianu M.C. Exploring UML refinement through unification. Proceedings of the UML'03 workshop on Critical Systems Development with UML, J. Jurjens, B. Rumpe, et al., editors - TUM-I0323, Technische Universitat Munchen, September 2003.
- [4] Cavalcanti A. and Naumann D. Simulation and Class Refinement for Java. In proceedings of ECOOP 2000 Workshop on Formal Techniques for Java Programs. (2000).
- [5] Davies J. and Crichton C. Concurrency and Refinement in the Unified Modeling Language. Electronic Notes in Theoretical Computer Science 70,3, Elsevier, 2002.
- [6] Derrick, J. and Boiten, E. Refinement in Z and Object-Z. Foundation and Advanced Applications. FACIT, Springer, 2001
- [7] Engels G., Küster J., Heckel R. and Groenewegen L. A Methodology for Specifying and Analyzing Consistency of Object Oriented Behavioral Models. Procs. of the IEEE Int. Conference on Foundation of Software Engineering. Vienna. (2001).
- [8] Kim, S. and Carrington, D., Formalizing the UML Class Diagrams using Object-Z, proceedings UML '99 Conference, Lecture Notes in Computer Science 1723 (1999).
- [9] Lano, K. The B Language and Method. FACIT. Springer, 1996.
- [10] Lano, K., Bicarregui, J., Formalizing the UML in Structured Temporal Theories, 2nd. ECOOP Workshop on Precise Behavioral Semantics, TUM-I9813, Technische U. Munchen (1998).
- [11] Ledang, Hung and Souquieres, Jeanine. Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B. Procs. of IEEE Asia-Pacific Software Engineering Conference 2002. December 4-6, 2002.
- [12] Liu, Z., Jifeng H., Li, X. Chen Y. Consistency and Refinement of UML Models. 3er Workshop on Consistency Problems in UML-based Software Development III, event of the UML Conference, 2004.
- [13] UML 2.0. The Unified Modeling Language Superstructure version 2.0 – OMG Final Adopted Specification. August 2003. <http://www.omg.org>.
- [14] OCL 2.0. OMG Final Adopted Specification. October 2003
- [15] Pons, C., Pérez, G., Giandini, R., Kutsche, Ralf-D. Understanding Refinement and Specialization in the UML. 2nd International Workshop on Managing Specialization/Generalization Hierarchies. In IEEE ASE 2003, Canada. Oct. 2003.
- [16] Pons, C. and Kutsche, R-D. Traceability across refinement steps in UML Modeling. Workshop in Software Model Engineering, 7th International Conference on the UML, Lisbon. October 11, 2004
- [17] Richters Mark and Gogolla Martin. OCL-Syntax, Semantics and Tools. in Advances in Object Modelling with the OCL. Lecture Notes in Computer Science number 2263. Springer. (2001).
- [18] Smith, Graeme. The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers. ISBN 0-7923-8684-1. (2000)
- [19] Van Der Straeten, R., Mens, T., Simmonds, J. and Jonckers, V. Using description logic to maintain consistency between UML-models. In Proc. 6th International Conference on the Unified Modeling Language. Lecture Notes in Computer Science number 2863. Springer. (2003).