Model Consistency in the Object Oriented Software Development Process

Gabriela A. Pérez
LIFIA - Laboratorio de Investigación y Formación en Informática Avanzada
Universidad Nacional de La Plata, Facultad de Informática
La Plata, Argentina, 1900
gperez@sol.info.unlp.edu.ar

ABSTRACT
Model Refinement is a relationship that relates two elements representing the same concept at different levels of abstraction. In UML, Refinement is described in an informal way.

To avoid wrong model interpretations, we study a formalization of the refinement relation. This work provides an enhancement to the UML metamodel specification.

Categories and Subject Descriptors
D.2 [**Software Engieneering**]: Requirements/Specifications, Methodologies (e.g., object-oriented, structured)

General Terms
Documentation, Design, Standardization, Languages.

Keywords
Object Oriented Analysis and Design, Unified Process, UML, Use Cases.

1. INTRODUCTION
A software development process, e.g. The Unified Process [5], is a set of activities needed to transform user's requirements into a software system. Modern software development processes are iterative and incremental, they repeat over a series of iterations making up the life cycle of a system. Each iteration takes place over time and it consists of one pass through the requirements, analysis, design, implementation and test activities, building a number of different artifacts (i.e. models). All these artifacts are not independent; they are semantically overlapping and together represent the system as a whole. Elements in one artifact have trace dependencies to other artifacts. On the other hand, due to the incremental nature of the process, each iteration results in an increment of artifacts built in previous iterations.

Model Refinement is carried out in different ways: for example: on the internal dimension (artifact-dimension), we can restrict the use case specification; On the vertical dimension (activity - dimension), analysis models are refinements of use case models, but on the other hand, on the horizontal dimension (iteration-dimension), models built in an iteration are usually refinements of models (of the same kind) built in previous iterations.

2. INTERNAL DIMENSION: USE CASE SPECIFICATION
Use cases can be specified in a number of ways, generally with natural language structured as a conversation between user and system. This conversation considers the normal action sequence and also alternative sequences. Each sequence represents a possible scenario of execution of the use case. Then, the complete description of a use case is composed by an scenario sequence.

In the UML metamodel, an action sequence is an instance of the ActionSequence metaclass, which is subclass of the Action metaclass. This fact generates some conflicting situations, like

* An ActionSequence could have arguments

* An ActionSequence could have an associated message

As a solution for these problems, we propose a new metamodel where the metaclass Action is sub classified with both CompositeAction and SimpleAction subclasses. A CompositeAction will be composed by actions, which could be acceded by the actionSequence association (this schema follows the pattern Composite [2]).

On the other hand, underlined actions[1] can appear in a use case conversation. The meaning of this is it will be refined.

It is interesting to be able to distinguish both concrete actions (i.e. atomic actions, that will not be refined) and abstract actions (i.e. actions that require a refinement). This situation neither is represented in the UML metamodel. We propose to sub classify a SimpleAction in both ConcreteAction and AbstractAction. ConcreteAction will be the superclass of the concrete actions that were defined in the metamodel until now (i.e. CallAction, CreateAction, etc.), while an AbstractAction will specify those actions that will be refined. These new metaclasses improve formality of the Use Case metamodel allowing for the definition of the Use Case refinement hierarchy, as we will see in next section.

3. HORIZONTAL DIMENSION: COLLABORATION REFINEMENT
Collaborations as well as use cases can be refined through subordinate collaborations, forming a refinement hierarchy. Each subordinate collaboration implements in more detail one part of

1 The notation used for conversation is based on [1]

OOPSLA'03, October 26–30, 2003, Anaheim, California, USA.
ACM 1-58113-751-6/03/0010.

the global functionality and can have its own sets of roles and interactions.

In order to formally specify the refinement relation between collaborations we define additional well formedness rules on the metaclass Collaboration.

In the case that a collaboration refines a message, we propose a standard format for such collaboration that allows a formal verification of well formedness of the refinement relation.

In order to understand this format it is necessary to consider that the sender of a message has less importance than the receiver of this message. The sender can be an instance of any ClassifierRole, however the receiver of the message is the responsible of the interpretation of that message. For this reason, we propose that the subordinate collaboration that describes the refinement of the original message begins this refinement with an instance of the ClassifierRole that received this message in the superordinate collaboration.

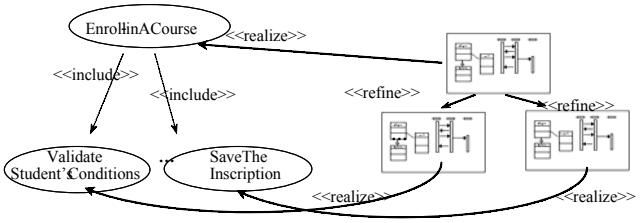
In addition, considering that the only messages that can be refined are those associated with CallActions, (since the other actions are atomic), we propose that the name of each subordinate collaboration will be formed by both the name of the Operation associated with that instance of CallAction (associated to the message), and the name of the base Classifier that contains such operation. In this way, examining the name of a subordinate collaboration we can associate it to at least one message in the superordinate collaboration, without ambiguity.

On the other hand, the ClassifierRole receiving of a refined message must maintain a relation with the ClassifierRole that sends the first message in each thread in the subordinate collaboration. It would be natural that in the refinement, we want to specify in more detail the ClassifierRole in charge of that behavior.

The ClassifierRole (first sender) in the subordinate collaboration should belong to the same generalization hierarchy as the original Classifier. In case that this Classifier is an interface the first sender must be a class that implements it.

4. CONSISTENCY CHECKING BETWEEN MODELS IN DIFFERENT DIMENSIONS

Since different models that are built during the software development process are related to each other along different dimensions, it is natural to perceive that interdependencies between dimensions could arise. For example, when a use case is realized by collaboration, it is expected that some specific relations hold between their respective subordinated elements, as we can see in the figure below.



Use case Refinement tree and Collaboration Refinement tree.

UML does not specify any constraint regulating these relationships. We define new well formedness rules -on the Collaboration metaclass- that allow us to verify consistency between two different refinement hierarchies, as follows:

self.representUseCase implies
(self.representedClassifier.hasIncluded implies
(self.representedClassifier.include.addition -> forAll
(subUseCase | self.usedCollaboration-> exists (subcol |
subcol.representedClassifier = subUseCase))))

This rule means that if a superordinate collaboration implements[2] a superordinate Use Case, then there must exist a subordinate collaboration implementing each subordinate Use Case.

5. CONCLUSIONS

In the UML specification document, several concepts are still described in an ambiguous, informal way. In previous works, we have analyzed other of these concepts ([4] [7] [8]). In this article we analyze the dependency relationship between models known as: "Model Refinement".

In order to avoid inconsistencies and wrong model interpretations, in this article we proposed, in first instance, a formalization of the Use Case specification, represented by a conversation between an actor and the system. The Use Case conversation did not have a representation in the UML metamodel. In second instance we proposed to formalize the refinement relations between model elements of the same kind. Finally, on top of these formalizations, we discussed refinement relations between models of different kind (use case models and collaboration models realizing them)

In particular, we defined well formedness rules in the OCL language, restricting Use Case specification as well as refinement hierarchy of both Use Cases and Collaborations.

The rules defined in this work form an enhancement of the UML metamodel specification. These rules should be used as a formal foundation for the construction of case tools performing consistency checking of models. Support offered by tools will improve the quality of the software development process.

6. REFERENCES

[1] Cockburn, Alistair. Writing Effective Use Cases. Addison-Wesley.

[2] Gamma, H. Design Patterns. Addison-Wesley, Professional Computing Series, 1995.

[3] Giandini,R, Pons, C and Baum,G.. An algebra for Use Cases in the UML. OOPSLA´00 Workshop on Behavioral Semantics, Minneapolis, USA, Oct. 2000.

[4] Giandini, R., Pons, C., Pérez,G. Use Case Refinements in the OO Software Development Process. Proceedings of CLEI 2002, ISBN 9974-7704-1-6, Uruguay. Nov. 2002.

[5] Jacobson, I..Booch, G Rumbaugh, J., The Unified Software Development Process, Addison Wesley. (1999)

[6] UML Specification. Version 1.4, Sept. 2001. Page 2-121

[7] Pons, Claudia, Pérez,Gabriela, Giandini, Roxana. Incremental Specialization vs. Overriding Specialization in the UML, IDEAS 2002. La Habana, Cuba., April 2002.

[8] Pérez,G., Giandini, R., Pons, C. Model Refinements in the Object Oriented Software Development Process ASSE. Argentina, ISSN 1666-1087 (61 - 73), Sept.2002.

[2] The attribute representedClassifier represents the Classifier (Class, Use Case, etc.) that the collaboration is realizing ([6])