Lecture Notes in Computer Science

The LNCS series reports state-of-the art results in computer science research, development, and education, at a high level and in both printed and electronic form. Enjoying tight cooperation with the R&D community, with numerous individuals, as well as with prestigious organizations and societies, LNCS has grown into the most comprehensive computer science research forum available.

The scope of LNCS, including its subseries LNAI and LNBI, spans the whole range of computer science and information technology including interdisciplinary topics in a variety of application fields. The type of material published traditionally includes

- proceedings (published in time for the respective conference)
- post-proceedings (consisting of thoroughly revised final full papers)
- research monographs (which may be based on outstanding PhD work, research projects, technical reports, etc.)

More recently, several color-cover sublines have been added featuring, beyond a collection of papers, various added-value components; these sublines include

- tutorials (textbook-like monographs or collections of lectures given at advanced courses)
- state-of-the-art surveys (offering complete and mediated coverage of a topic)
- hot topics (introducing emergent topics to the broader community)

In parallel to the printed book, each new volume is published electronically in LNCS Online.

Detailed information on LNCS can be found at
http://www.springeronline.com

Proposals for publication should be sent to

LNCS Editorial, Tiergartenstr. 17, 69121 Heidelberg, Germany

E-mail: lncs@springer.de

ISSN 0302-9743

Lecture Notes in Computer Science
LNCS LNAI LNBI

springeronline.com

Baar · Strohmeier
Moreira · Mellor (Eds.)

LNCS 3273

«UML» 2004 – The Unified Modeling Language

«UML» 2004

Thomas Baar Alfred Strohmeier
Ana Moreira Stephen J. Mellor (Eds.)

«UML» 2004 – The Unified Modeling Language

Modeling Languages and Applications

7th International Conference
Lisbon, Portugal, October 2004
Proceedings

Springer

Lecture Notes in Computer Science 3273

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Thomas Baar Alfred Strohmeier
Ana Moreira Stephen J. Mellor (Eds.)

«UML» 2004 – The Unified Modeling Language

Modeling Languages and Applications

7th International Conference
Lisbon, Portugal, October 11-15, 2004
Proceedings

Springer

Volume Editors

Thomas Baar
Alfred Strohmeier
Swiss Federal Institute of Technology Lausanne (EPFL)
Software Engineering Laboratory
1015 Lausanne, Switzerland
E-mail: {thomas.baar,alfred.strohmeier}@epfl.ch

Ana Moreira
Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia
Departamento de Informática
Quinta da Torre, 2829-516 Caparica, Portugal
E-mail: amm@di.fct.unl.pt

Stephen J. Mellor
Mentor Graphics, Corp., Accelerated Technology
Suite 365, 7400 N. Oracle Road, Tucson, AZ 85704, USA
E-mail: Stephen_Mellor@Mentor.com

Library of Congress Control Number: 2004112978

CR Subject Classification (1998): D.2, D.3, K.6, I.6

ISSN 0302-9743
ISBN 3-540-23307-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 11329060 06/3142 5 4 3 2 1 0

Preface

This volume comprises the final versions of the technical papers presented at the ≪UML≫ 2004 conference held in Lisbon (Portugal), October 11–15, 2004.

≪UML≫ 2004 was the seventh conference in a series of annual ≪UML≫ conferences. The series started in 1998 and was located at Mulhouse (France); the following years saw the conference move to Fort Collins (USA) in 1999, York (UK) in 2000, Toronto (Canada) in 2001, Dresden (Germany) in 2002, San Francisco (USA) in 2003, and now Lisbon (Portugal) in 2004. During this time, the ≪UML≫ conference became one of the leading conferences in the area of object-oriented modeling. While in the first years the focus of the conference was on the scientific investigation of the Unified Modeling Language (UML), which had just been adopted by the Object Management Group (OMG) at the time, the focus has changed in recent years to innovations in techniques such as metamodeling, model transformations, model validation and verification, aspect orientation, and beyond. Many recent research activities have been especially stimulated by the Model Driven Architecture (MDA) initiative, started in 2000 by the OMG. The goal of MDA is the definition of a framework to enable the development of software purely based on models. In order to reflect the changes of recent years, the conference series ≪UML≫ will be continued, from 2005 onwards, under the name MODELS (MOdel Driven Engineering, Languages and Systems).

The call for papers for ≪UML≫ 2004 encouraged authors around the world to submit 157 abstracts and 135 technical papers. Each submission was reviewed by at least three referees, in most cases by four. Based on the reviews, the whole program committee discussed in four rounds the submissions' quality, leading to the selection of 30 submissions (26 research papers, 4 experience reports) for publication. In addition, the program committee selected one paper for the *Best Paper Award ≪UML≫ 2004*. After a detailed discussion of selected candidates the committee came to the conclusion that the paper by Alexandre Correa, Cláudia Werner (Brazil), "Applying Refactoring Techniques to UML/OCL Models", deserved the award. Congratulations to the authors!

For managing the review process, the free version of Cyberchair (`http://www.cyberchair.org`) was used. We are grateful to its author Richard van de Stadt who also helped with advice. We also want to take the opportunity to express our greatest gratitude to Arnaud di Clemente whose work on the technical side was invaluable for managing the review process and preparing the conference proceedings.

Besides the presentation of technical papers in 10 sessions, the scientific program of ≪UML≫ 2004 included 3 keynote talks, "Generative Software Development", given by Krzysztof Czarnecki (University of Waterloo), "Goals, Viewpoints, and Components — an MDA Perspective", given by Desmond D'Souza (Kinetium), and "Putting Change at the Center of the Software Process", given

by Oscar Nierstrasz (University of Bern), 12 workshops, including a doctoral symposium, 6 tutorials, and a special track with industry papers. In addition to this proceedings, a postconference book entitled *≪UML≫ 2004 Satellite Activities* was published by Springer as LNCS volume 3297. This book includes the papers of the industry track, summaries of the workshops, tool papers and poster papers.

We are glad to express our gratitude to all persons and organizations who were involved in the organization of the conference: to the sponsors and supporters for the financial, organizational, and moral aid, to the reviewers for their dedication in writing reports and contributing to the discussion, and to the members of the local organization committee for their incredible work in coordinating all activities and making the local arrangements.

July 2004

Thomas Baar
Alfred Strohmeier
Ana Moreira
Stephen J. Mellor

Organization

Executive Committee

General Chair	Stephen J. Mellor (Mentor Graphics, USA)
Conference Chair	Ana Moreira (New University of Lisbon, Portugal)
Program Co-chairs	Thomas Baar (EPFL, Switzerland) Alfred Strohmeier (EPFL, Switzerland)
Industry Track Chair	Bran Selic (IBM Rational Software, Canada)
Tutorial Chair	Ezra K. Mugisa (University of the West Indies at Mona, Jamaica)
Workshop Chair	Ambrosio Toval (University of Murcia, Spain)
Panel Chair	Jon Whittle (NASA Ames Research Center, USA)
Poster Chair	Nuno Jardim Nunes (University of Madeira, Portugal)

Organizing Team

Publicity Chairs	João Araújo (New University of Lisbon, Portugal) Geri Georg (Colorado State University, USA)
Local Arrangements Chair	Isabel Sofia Brito (Politécnico de Beja, Portugal)
Tools Exhibition Chair	Alberto Silva (Technical University of Lisbon, Portugal)
Local Sponsors Chair	Fernando Brito e Abreu (New University of Lisbon, Portugal)
Web Chair	Miguel Goulão (New University of Lisbon, Portugal)

Program Committee

Mehmet Aksit (The Netherlands)
Omar Aldawud (USA)
Colin Atkinson (Germany)
Doo-Hwan Bae (Korea)
Jean Bézivin (France)
Marko Boger (Germany)
Ruth Breu (Austria)
Jean-Michel Bruel (France)
David Bustard (UK)
Alessandra Cavarra (UK)
Betty Cheng (USA)
Siobhán Clarke (Ireland)
John Daniels (UK)
Stéphane Ducasse (Switzerland)
Gregor Engels (Germany)
Andy Evans (UK)
Robert France (USA)
Sébastien Gérard (France)
Martin Gogolla (Germany)
Jeff Gray (USA)
Constance Heitmeyer (USA)
Brian Henderson-Sellers (Australia)
Heinrich Hussmann (Germany)
Pankaj Jalote (India)

Stuart Kent (UK)
Jörg Kienzle (Canada)
Haim Kilov (USA)
Philippe Kruchten (Canada)
Tim Lethbridge (Canada)
Richard Mitchell (USA)
Hiroshi Miyazaki (Japan)
Pierre-Alain Muller (France)
Ileana Ober (France)
Gunnar Overgaard (Sweden)
Ernesto Pimentel Sanchez (Spain)
Gianna Reggio (Italy)
Laurent Rioux (France)
Bernhard Rumpe (Germany)
Peter H. Schmitt (Germany)
Andy Schürr (Germany)
Bran Selic (Canada)
R.K. Shyamasundar (India)
Keng Siau (USA)
Jos Warmer (The Netherlands)
Alain Wegmann (Switzerland)
Jon Whittle (USA)

Additional Reviewers

Aditya Agrawal
Muhammad Alam
Carsten Amelunxen
Gabriela Arévalo
Egidio Astesiano
Richard Atterer
Pavel Balabko
Elisa Baniassad
Alexandre Bergel
Lodewijk Bergmans
Egon Börger
Marius Bozga
Richard Bubel
Fabian Büttner
Robert D. Busser
Maura Cerioli
Alexey Cherchago
Joanna Chimiak-Opoka
Olivier Constant
Steve Cook
James Davis
Gregory Defombelle
Birgit Demuth
Min Deng
Ludovic Depitre
Manuel Díaz
Cormac Driver
Hubert Dubois
Francisco J. Durán
Cristian Ene
Brit Engel
Alexander Förster
Per Fragemann
Markus Gälli
Geri Georg
Cesar Gonzalez-Perez
Orla Greevy
Jiang Guo
Michael Hafner
Nabil Hameurlain
Michel Hassenforder
Jan Hendrik Hausmann
Reiko Heckel
Karsten Hölscher
Hardi Hungar
Andrew Jackson
Bernhard Josko
Frédéric Jouault
Andreas Kanzlers
Stephen Kelvin
Dae-Kyoo Kim
Alexander Knapp
Alexander Königs
Matthias Köster
Sascha Konrad
Holger Krahn
Jochen Küster
Juliana Küster-Filipe
Ivan Kurtev
Pierre Laforcade
Benoit Langlois
Lam-Son Lê
Alexander Lechner
Yuehua Lin
Arne Lindow
Chang Liu
Sten Loecher
Marc Lohmann
Shiu Lun Tsang
Viviana Mascardi
Girish Maskeri
Dan Matheson
Wei Monin
Stefan Müller
Andronikos Nedos
Thierry Nodenot
Joost Noppen
Iulian Ober
Jean-Marc Perronne
Andreas Pleuss
Erik Poll
Tadinada V. Prabhakar
Birgit Prammer
Raghu Reddy
Gil Regev
Wolfgang Reisig
Arend Rensink
Robert Rist
Tobias Rötschke
Andreas Roth

Suman Roychoudhury
Enrico Rukzio
Irina Rychkova
Goiuria Sagardui
Paul Sammut
Tim Schattkowsky
Steffen Schlager
Jean-Marc Seigneur
Magdy Serour
Vibhu Sharma
Andrew Simpson
Randy Smith
Jonathan Sprinkle
Thomas Sproesser
Dominik Stein
Ryan Stephenson
Philippe Studer
Thorsten Sturm
Bedir Tekinerdoğan
Bernard Thirion
Stavros Tripakis
Dinh-Trong Trung
Ambrosio Toval
Antonio Vallecillo
Klaas van den Berg
Pim van den Broek
Jesco von Voss
Barbara Weber
Reinhard Wilhelm
James Willans
Alan Cameron Wills
Mariemma I. Yagüe
Zhenxiao Yang
Jing Zhang
Paul Ziemann
Steffen Zschaler

Sponsors

SINFIC
http://www.sinfic.pt

Springer
http://www.springeronline.com

Mentor Graphics
http://www.mentor.com

IBM France
http://www.ibm.com/fr

Supporters

ACM Special Interest Group
on Software Engineering
http://www.acm.org

IEEE Computer Society
http://www.ieee.com

New University of Lisbon
http://di.fct.unl.pt

Turismo de Lisboa
http://www.tourismlisbon.com

Object Management Group,
http://www.omg.org

Table of Contents

OCL

Model Transformation

Verification and Model Consistency

Security

Methodology

Workshops and Tutorials

Extending OCL for Secure Database Development

Eduardo Fernández-Medina and Mario Piattini

Escuela Superior de Informática. Universidad de Castilla-La Mancha
Paseo de la Universidad, 4. 13071, Ciudad Real (Spain)
{Eduardo.FdezMedina, Mario.Piattini}@uclm.es

Abstract. The Model Driven Architecture (MDA) is becoming an important aspect of software development, since it considers languages and models that can represent an information system at different abstraction levels, and makes it possible a coherent transformation of the system from the domain context into the machine context. In this paper, we present the Object Security Constraint Language V.2. (OSCL2), which is based on the well-known Object Constraint Language V.2. (OCL) of the Unified Modeling Language (UML), and which needs an extension of the UML[1] metamodel. This language is defined to be used in secure database development process, incorporating security information and constraints in a Platform Independent Model (UML class model). This security information and constraints are then translated into a Platform Specific Model (multilevel relational model). Finally, they are implemented in a particular Database Management System (DBMS), such as Oracle*9i* Label Security. These transformations can be done automatically or semi-automatically using OSCL2 compilers.

Keywords: OCL, security constraints, multilevel databases, UML, confidentiality.

1. Introduction

Organizations depend increasingly on IS, which rely upon large databases, and these databases need increasingly more quality and security. Indeed, the very survival of the organization depends on the correct management, security and confidentiality of this information [10].

As some authors remarked [9, 13], information security is a serious requirement which must be considered carefully, not as an isolated aspect, but as an element present in all stages of the development life cycle, from the requirement analysis to implementation and maintenance. For this purpose, different ideas for integrating security in the system development process have been proposed [16], but they only considered information security from a cryptographic point of view. Chung et al. also insist on integrating security requirements in the design, by providing designers with models specifying security aspects, but they do not deal with database specific issues [3]. There are a few proposals that try to integrate security into conceptual modeling such as the Semantic Data Model for Security [27] and the Multilevel Object Modeling Technique [23], but they have not been very spread. One more recent proposal is UMLSec [18] where UML is extended to develop secure systems. This approach is very interesting, but it -again- only deals with IS in general, whilst conceptual and logical databases design, and their implementation are not considered. All these proposals make important contributions that try to solve the problem of developing secure information (sometimes also database) systems, but they are not Model Driven Architecture (MDA) compliant [21]. In fact, they do not define neither development methodologies, nor several models at different conceptual levels. Nevertheless, a methodology and a set of models have been proposed [12] in order to design secure databases, and implement them with Oracle*9i* Label Security (OLS) [22]. This approach is important, because it considers security aspects in all stages of the development process, from requirement gathering to implementation, and it is also based on UML [1]. Together with the methodology mentioned above, a preliminary version of the Object Security Constraint Language (OSCL) has been proposed [25]. This language is based on the Object Constraint Language (OCL) [28, 29] of UML, and it allows us to specify security constraints in the conceptual and logical database design process, and to implement these constraints in a concrete database management system (DBMS), Oracle*9i* Label Security.

OCL is a precise textual language for describing constraints in object-oriented models. This language complements diagrammatic notations in modeling object-oriented systems, defining constraints which can not be described using the standard UML diagrammatic notation. The most important elements that are defined in OCL are invariants (constraints that establish a condition that must be always fulfilled by all the class instances, types and interfaces), preconditions (conditions related to class operations and which must be fulfilled when an operation is executed), and postconditions (conditions related to operations that generally specify the result of operations and which must be considered them true only at the moment the operation finishes).

In the past few years, some proposals to extend OCL to include new properties have been presented. In [14], an OCL extension in order to define real-time constraints is stated. Also, in [17], an approach for incorporating time-based constraints within OCL is described. Moreover, in [30], OCL is extended with temporal operators to formulate temporal constraints, and [5, 19, 20] present an extension of OCL in order to introduce dynamic semantics by adding an action clause to the language. OCL is being used and extended in order to model relational databases with UML [8], or even to model operation contracts to preserve low coupling [24]. Nevertheless, none of these proposals allows us to specify security constraints in conceptual database models. We have only found a work [2] that uses OCL to model software reliability constraints, but it does not match perfectly with the problem we are considering.

In this paper, we define some UML metamodel extensions to be used together with the second version of OSCL language, taking into account the new properties of OCL Version 2.0 [29]. OSCL allows us to define security information in the database conceptual model (through the UML class model) as well as security constraints that can be implemented in a concrete secure DBMS. Recently, several DBMS are including modules in order to manage and implement secure databases, such as OLS [22] and DB2 Universal Database (UDB) [6, 7]. In this work, according to MDA [21], we have considered the UML class model as Platform Independent Model (PIM), the multilevel relational model [12] as Platform Specific Model (PSM), and OLS as DBMS to implement the secure database. However, following the properties of MDA, it could

[1] We based our extension on the UML 1.5 as this is the current accepted standard.

T. Baar et al. (Eds.): UML 2004, LNCS 3273, pp. 380-394, 2004.

be perfectly valid to consider another database logical paradigm and another secure DBMS.

The remainder of this paper is structured as follows: Section 2 briefly introduces the methodology we proposed for developing secure databases. Section 3 states OSCL2 language and necessary UML metamodel extensions. Section 4 shows an example where the new constraint language is used, and Section 5 presents our main conclusions and introduces our future work.

2. MDA-compliant Secure Database Development Methodology

The goal of the methodology presented in [12] is to be able to develop databases, but classifying the information in order to define what properties have to own the user to be entitled to access the information. In order to fulfill the previous goal, we need to carry out the following activities:

1. To precisely define the organization of the users that will have access to the database. We can define a precise level of granularity considering three ways of organizing the users: *Security hierarchical levels* (which indicate the level of accreditation the user owns), *Compartments* of users (which indicate a horizontal classification of users), and *user Roles* (which indicate a hierarchical organization of users according to the roles or responsibilities of users within the organization).
2. To classify the information within the conceptual database model. For each element of the model (classes, attributes and associations), we can define its security information, which is a tuple composed of a sequence of security levels, a set of user compartments, and a set of user roles. This information indicates the security properties that the users have to own to be able to access the information.
3. To enforce the mandatory access control [26]. The secure DBMS is in charge of ensuring the enforcement of the mandatory access control, and the security information that has been specified in the database conceptual model has to be kept.

These activities are embedded in a methodology for developing secure databases [12]. The general structure of this methodology can be observed in Fig. 1. The *requirements gathering*, *database analysis*, and *multilevel relational logical design* stages, allow us to build general (conceptual and logical) models of the secure database, and finally, the *specific logical design* stage adapts those general models to the particularities of Oracle9i Label Security.

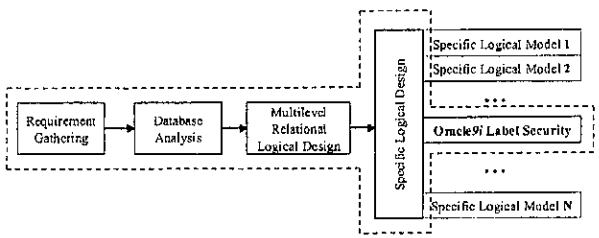

Fig. 1. Secure database development methodology stages

This methodology is MDA-compliant because it is open to integrate, after the database analysis stage (where the conceptual model is created), another logical design stage in order to create another secure logical model of the database (object-oriented, or object-relational). It is also open to integrate, after the multilevel relational logical design, other specific logical design in order to implement the secure database by other DBMS (See Fig. 2.).

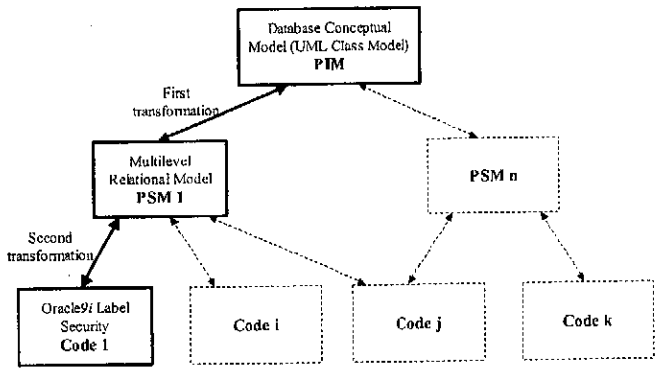

Fig. 2. MDA-compliant models

OSCL2 is integrated into the database analysis, where it helps in both the definition of security information of the conceptual database model elements and the definition of security constraints of the model. This security information and constraints are adapted to the logical model in the multilevel relational logical design, and finally, they are implemented in the specific logical design.

3. Object Security Constraint Language Version 2

According to [4], an extension to UML begins with a brief description and then it lists and describes all of the stereotypes, tagged values, and constraints of this extension. In addition to these elements, an extension contains a set of well-formedness rules. These rules are used to determine whether a model is semantically self-consistent. According to this quote, we define our OCL extension following the schema composed of these elements: Description (a little description of the extension in natural language), prerequisite extensions (they indicate whether the current extension needs the existence of previous extensions), stereotypes/tagged values (the definition of the stereotypes and/or tagged values), well-formedness rules (the static semantics of the metaclasses are defined as a set of invariants defined by means of OCL expressions), and comments (any additional comment, decision or example, usually written in natural language). For the definition of the stereotypes, we follow the structure suggested in [15], which is composed of a name, the base metaclass, the description, the tagged values and a list of constraints defined by means of OCL. For the definition of tagged

values, the type of the tagged values, the multiplicity, the description, and the default value are defined.

3.1. Description

The extension we present in this section is not only an extension of the OCL, but also the definition of a new framework that needs the definition of some new data types, and that allows us to specify both security information of the different elements of the database conceptual model (through tagged values), and dynamic and static security constraints of these elements. These constraints, allow us to dynamically define the security information of instances of classes and its attributes, depending on the value of attributes of these instances. Moreover, in this framework, a set of inherent constraints are specified in order to define the well-formedness rules regarding the values of these tagged values.

This framework allows us to represent all the security information (both tagged values and constraints) in the same model and in the same diagrams that describe the rest of the system.

The security information that we will embed in the database conceptual model will be security levels, user compartments, and user roles. This information indicates the security properties that the users have to own to be able to access information. Fig. 3 shows an extension of the pattern that has been presented in [11], that explains the mandatory access control.

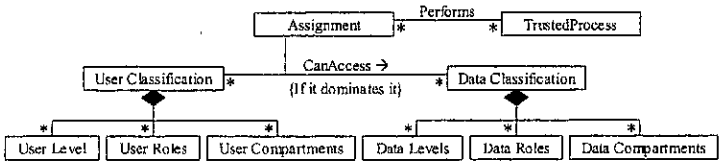

Fig. 3. Class model for the mandatory access control

3.2. Prerequisite Extensions

First of all, we need the definition of some new data types to be used in the tagged values definitions (see Fig. 4). The *Level* type will be the ordered enumeration composed by all security levels that have been considered (these values, are generally *unclassified*, *confidential*, *secret* and *top secret*, but they could be different). The *Levels* type will be an interval of levels composed by a lower level and an upper level. The *Role* type will represent the hierarchy of user roles that can be defined for the organization (in this case, we consider there is no multiple inheritance). The *Compartment* type is the enumeration composed by all user compartments that have been considered for the organization. All this information has to be defined for each database, depending on its confidentiality properties, and on the number of users and complexity of the organization which the database will be operative in.

Finally, we need some syntactic definitions that are not considered in the standard OCL. Particularly, in addition to *Set*, *OrderedSet*, *Bag* and *Sequence*, we need the following collection type:

- *Tree*: It is a collection that contains a root and a sequence of trees. A Tree can be empty.

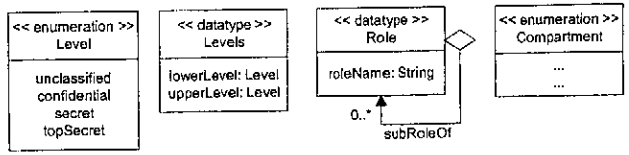

Fig. 4. New data types

Despite trees could be specified by complex OCL structures (see Fig. 5. (a) that shows the collections constant definition of the tree that is shown in Fig. 5. (b)), for the sake of simplicity, we prefer to use a new collection type.

```
Tree {
  'Employee',
  Sequence { Tree { 'System Administrator' , Sequence {} },
             Tree { 'Operator' , Sequence {} }
           }
}
```

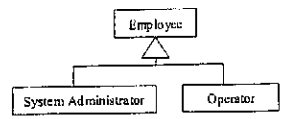

Fig. 5. (a) Example of Tree collection (b) Example of Tree

All the standard operations in this collection are applicable, but we need to define two new operations. Table 1 shows all these operations.

Table 1. Standard and new operations applicable to the *Tree* collection

Operation	Description
count (object)	The number of occurrences of the object in the tree
excludes(object)	True if the object is not an element of the collection
excludesAll(collection)	True if all elements of the parameter are not present in the current collection. This operation is applicable if the collection parameter type is Tree.
Includes(object)	True if the object is an element of the collection
includesAll(collection)	True if all elements of the parameter collection are present in the current collection. This operation is applicable if the collection parameter type is Tree.
IsEmpty()	True if the collection contains no elements
notEmpty()	True if the collection contains one or more elements
Size()	The number of elements in the collection
Sum()	The addition of all elements in the collection. The element must be of a type supporting addition
Root()	**The root of the tree**
Subtree(n)	**The subtree n of the sequence of subtrees of a tree**

3.3. Tagged Values

In this extension, the definition of several types of tagged values is necessary. We need to define tagged values for the conceptual database model, classes, attributes and associations.

Table 2. Tagged values of the model, classes and attributes

Tagged Values of the Model		
Name	Type	Description
Classes	Set(OclType)	It specifies all classes of the model. This new tagged value is useful in order to navigate over all classes of the model
securityLevels	Sequence (Levels)	It specifies all security levels (ordered from less to more restrictive) that can be used by the elements of the model
SecurityRoles	Role	It specifies the hierarchical role structure that has been defined for the organization. This type will be managed as a tree
security-Compartments	Set (Compartment)	It specifies the set of compartments that have been defined for the organization
Tagged Values of the Class		
Name	Type	Description
Attributes	Set(OclType)	It specifies all attributes of the model. This new tagged value is useful in order to navigate over all attributes of a class
associations-End	Set(OclType)	It specifies all associations in which a class is involved. This new tagged value is useful in order to navigate over all associations of a class
securityLevels	Levels	It specifies the interval of possible security level values, that an instance of this class can receive. If the upper and lower security levels are similar, all instances will have the same security level. Otherwise, the concrete instance security level will be defined according to a security constraint
SecurityRoles	Set(Role)	It specifies a set of user roles. Each role is the root of a subtree of the general user role hierarchy defined for the organization. All instances of this class can have the same user roles, or maybe subtrees of the roles that have been defined for the class. A security constraint can decide the user roles for each instance according to the value of some attribute of the instance
security-Compartments	Set (Compartment)	It specifies a set of compartments. All instances of this class can have the same user compartments, or a subset of them. A security constraint can decide the user compartments for each instance according to the value of some attribute of the instance
Tagged Values of the Attribute		
Name	Type	Description
securityLevels	Levels	It specifies the interval of possible security level values that an attribute of an instance can receive. If the upper and lower security levels are similar, all instances will have the same security level. Otherwise, the concrete instance security level will be defined according to a security constraint
SecurityRoles	Set(Role)	It specifies a set of user roles for this attribute. Each role is the root of a subtree of the general user role hierarchy defined for the organization. All instances for this attribute can have the same user roles, or maybe subtrees of the roles that have been defined for the attribute. A security constraint can decide the user roles for each instance according to the value of the attribute of the instance
security-Compartments	Set (Compartment)	It specifies the set compartments for an attribute. For this attribute, all instances can have the same user compartments, or a subset of them. A security constraint can decide the user compartments for each instance according to the value of this attribute

In Table 2 and 3 the tagged values of all elements of this extension are shown. The multiplicity of all tagged values is 1. All default values of security tagged values of the model are empty collections. On the other hand, the default value of security tagged values for each class is the less restrictive (the lower security level, the security role hierarchy that has been defined for the model, and the empty set of compartments). The default value of the security tagged values for attributes is inherited from the class they belong to. For associations, the security information is the most restrictive from all classes that are involved in this association (the security levels of the most restrictive class, the intersection of the security roles of the classes, and the intersection of the security compartments of the classes).

Table 3. Tagged values of associationEnd and instance

Tagged Values of the AssociationEnd		
Name	Type	Description
securityLevels	Levels	It specifies the interval of possible security level values that an association between two instances can receive. If the upper and lower security levels are similar, all links will have the same security level. Otherwise, the concrete link security level will be defined according to a security constraint
securityRoles	Set(Role)	It specifies a set of user roles for this association. Each role is the root of a subtree of the general user role hierarchy defined for the organization. All links of this association can have the same user roles, or maybe subtrees of the roles that have been defined for the association. A security constraint can decide the user roles for each link according to the value of the attribute of the instances
security-Compartments	Set (Compartment)	It specifies the set compartments for an association. For this association, all links can have the same user compartments, or a subset of them. A security constraint can decide the user compartments for each link according to the value of the attribute of the instances
Tagged Values of the Instance		
Name	Type	Description
securityLevel	Level	It specifies the security level of an instance
securityRoles	Set(Role)	It specifies a set of user roles for this instance. Each role is a subtree of the general user role hierarchy defined for the organization.
security-Compartments	Set (Compartment)	It specifies the set compartments for an instance

3.4. Stereotypes

Once we have all these tagged values available, we could specify security constraints in a class diagram, but all depending on the value of attributes and specified tagged values. In this extension, we need to define some stereotypes in order to specify other types of security constraints (see Table 4).

We define the *UserProfile* stereotype because it could be necessary to specify constraints depending on particular information of a user or a group of users, for instance, depending of the user citizenship, age, etc. We define the *Exception* stereotype because of the necessity of specifying different constraints which permit or deny access to information depending on information different from the security levels, roles or compartments. We also define the *Log* stereotype that can help us to identify and specify classes with special requirements of auditing.

For the sake of readability, the stereotypes *Exception* and *Log* can be graphically specified outside of the class.

Table 4. Extension stereotypes

Name	**UserProfile**
Base class	Class
Description	Classes of this stereotype contain all the properties that the systems manage from users

Constraints	- This class has no associations to other classes
	`self.AssociationsEnd.size()=0`
	- There is no more than one class of this type
	`Context Model`
	`Inv self.classes->forAll(oclIsTypeOf(UserProfile))->size()<=1`
	- The name of a class of this stereotype will be "UserProfile"
	`self.className="UserProfile"`

Tagged Values	None
Name	**Log**
Base class	Class
Description	Classes of this stereotype have special security constraints: the information of all accesses to this class has to be recorded in a log file for future audit
Constraints	None
Tagged Values	- Type - Type: {all, frustratedAttempt, successfullAccess} - Multiplicity: 1 - Description: It indicates whether the access has to be recorded: all accesses, only accesses that have been frustrated, or only successful accesses - Condition - Type: OCLExpression - Multiplicity: 0..* - Description: It indicates whether the access has to be recorded
Name	**Exception**
Base class	Class
Description	Classes of this stereotype will have a special security constraint. This constraint specifies an exception by which a user or a set of users (depending on a condition) can (or cannot) access to the corresponding class, independently of the its security information
Constraints	None
Tagged Values	- Sign - Type: {+,-} - Multiplicity: 1 - Description: It indicates if the exception permit (+) or deny (-) access to instances of this class to a user or a group of users - Privileges - Type: {read, insert, delete, update, all} - Multiplicity: 1..* - Description: It indicates the privileges the user specified in this constrain can receive or remove. - Condition - Type: OCLExpression - Multiplicity: 0..* - Description: It specifies the condition that users have to fulfill to be affected by this exception

3.5. Well-Formedness Rules

We can identify and specify with complex OSCL2 constraints many well-formedness rules. These rules are grouped as follows:

- Correct value of the tagged values:
  - The security levels defined for each class of the model, for each attribute, and for each association, have to belong to the sequence of security levels that has been defined for the model.

```
context Model
inv self.classes-> forAll(c | self.securityLevels -> includesAll
(subSequence[2](c.securityLevels.lowerLevel, c.securityLevels.upperLevel))
inv self.classes-> forAll(c | c.attributes-> forAll(a |
self.securityLevels-> includesAll(subSequence(a.securityLevels.lowerLevel,
a.securityLevels.upperLevel)))
inv self.classes-> forAll(c | c.associationsEnd-> forAll(a |
self.securityLevels-> includesAll(subSequence(a.securityLevels.lowerLevel,
a.securityLevels.upperLevel)))
```

  - The set of user roles defined for each class, attribute, and association of the model has to be a subtree of the roles tree that has been defined for the model.

```
context Model
inv self.classes-> forAll(c | c.Roles-> forAll( r | self.Role-
>includesAll(r)))
inv self.classes-> forAll(c | c.attributes-> forAll(a | a.Roles-> forAll (r
| self.Role-> includesAll(r))))
inv self.classes-> forAll(c | c.associationsEnd-> forAll(a | a.Roles->
forAll (r | self.Role-> includesAll(r))))
```

  - The set of user compartments defined for each class, attribute and association of the model has to be a subset of the model compartments.

```
context Model
inv self.classes-> forAll(c | c.Compartments-> forAll( comp |
self.Compartments->includes(comp)))
inv self.classes-> forAll(c | c.attributes-> forAll(a | a.Compartments->
forAll (comp | self.Compartments-> includes(comp))))
inv self.classes-> forAll(c | c.associationsEnd-> forAll(a |
a.Compartments-> forAll (comp | self.Compartments-> includes(comp))))
```

- The security information of instances:
  - The security level of the instance of a class has to be included in the interval of security levels that has been defined for the class. The same rule is applicable to the instances of attributes and links.

```
context Model
inv self.classes-> forAll(c | c.allInstances -> forAll (i |
self.securityLevels-> subSequence(c.securityLevels.lowerLevel,
c.securityLevels.upperLevel)-> includes(i.securityLevel)))
```

  - The user roles of an instance of a class have to be subtress of the roles trees that have been defined for the class. The same rule is applicable to the instance of attributes and links.

```
context Model
inv self.classes-> forAll(c | c.allInstances -> forAll (i |
c.securityRoles-> includesAll(i.securityRoles)))
```

  - The user compartments of an instance of a class have to be a subset of the compartments that have been defined for the class. The same rule is applicable to the instance of attributes and links.

```
context Model
inv self.classes-> forAll(c | c.allInstances -> forAll (i |
i.securityCompartments-> includesAll(i.securityCompartments)))
```

- Relationship between the security information of classes, and its attributes:
  - The security levels defined for an attribute have to be equal or more restricted than the security levels defined for its class. The same rule is applicable to the role hierarchies and user compartments.

[2] The type of the arguments of *subSequence* collection is integer, but for the sake of readability, we consider that the arguments can be elements of the subSequence. The correct expression should be subSequence(self.securityLevels->indexOf(c.securityLevels.lowerLevel), self.securityLevels->indexOf(c.securityLevels.upperLevel). We consider this simplification in all uses of *subSequence* operation.

```
context Model
inv self.classes-> forAll(c | c.attributes-> forAll(a |
self.securityLevels-> subSequence(c.securityLevels.lowerLevel,
a.securityLevels.upperLevel)-> includesAll(self.securityLevels->
subSequence(a.securityLevels.lowerLevel,a.securityLevels.upperLevel)))
inv self.classes-> forAll(c | c.attributes-> forAll (a | c.securityRoles->
includesAll(a.securityRoles)))
inv self.classes-> forAll (c | c.attributes-> forAll (a |
c.securityCompartments-> includesAll(a.securityCompartments)))
```

- Relationship between the security information of classes, and its associations:
  - The security levels defined for an association between two classes have to be equal or more restricted than the security level of these classes. The same rule is applicable to the role hierarchies and user compartments

```
context Model
inv self.classes-> forAll(c | c.associationsEnd-> forAll(a | a.participant-
> forAll (ca | self.securityLevels->
subSequence(ca.securityLevels.lowerLevel, a.securityLevels.upperLevel)->
includesAll(self.securityLevels-> subSequence
(a.securityLevels.lowerLevel,a.securityLevels.upperLevel)))
inv self.classes-> forAll(c | c.associationsEnd-> forAll (a |
a.participant-> forAll (ca | ca.securityRoles->
includesAll(a.securityRoles)))
inv self.classes-> forAll (c | c.associationsEnd-> forAll (a |
a.participant-> forAll (ca | ca.securityCompartments->
includesAll(a.securityCompartments)))
```

- Generalization hierarchies:
  - The rule we consider for this type of relationships is as follows: The security level of the subclasses has to be equal or more restrictive than the security level of the superclass. This rule is applicable to user roles and compartments.

```
context Model
inv self.classes-> forAll(c | c.subClasses-> forAll(s |
self.securityLevels-> subSequence(c.securityLevels.lowerLevel,
s.securityLevels.upperLevel)-> includesAll(self.securityLevels->
subSequence (s.securityLevels.lowerLevel,s.securityLevels.upperLevel))
inv self.classes-> forAll(c | c.subClasses-> forAll (s | c.securityRoles->
includesAll(s.securityRoles))
inv self.classes-> forAll (c | c.subClasses-> forAll (s |
c.securityCompartments-> includesAll(s.securityCompartments)))
```

3.6. Comments

Many of the previous constraints are very intuitive, hence, we have to ensure their fulfillment since, in other case the system would be inconsistent. Additionally to these inherent constraints, the designer can specify security constraints with OSCL2. If the security information of a class or an attribute depends on the value of an attribute of an instance, it can be expressed as an OSCL2 expression (see Section 4).

4. Example Where OSCL2 Is Used

In this section, we apply the extension we are dealing with for the conceptual design of a secure database in the context of a typical health-care information system. We have only selected two classes in order to focus the example on the specification of security information and constraints. Fig. 6 (a) shows the simplified hierarchy of user roles of the system, and Fig. 6 (b) shows the security levels that have been defined. In this example, compartments have not been defined.

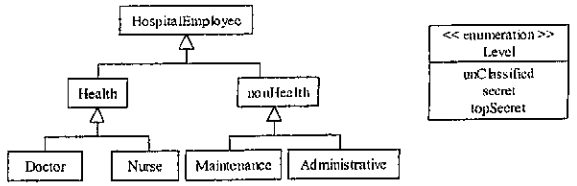

Fig. 6. (a) User roles hierarchy (b) Security levels

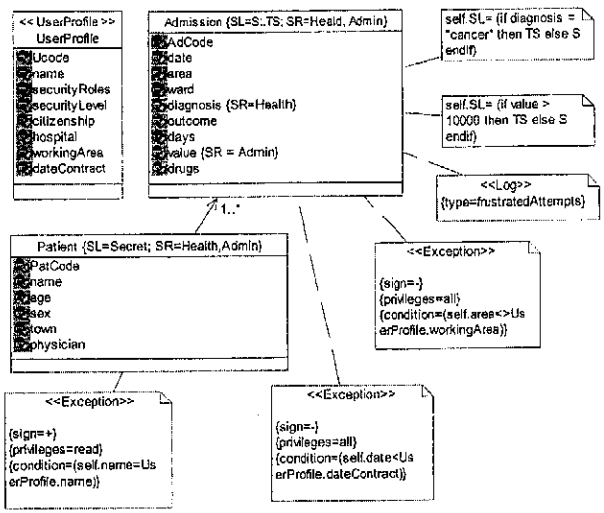

Fig. 7. Example of class diagram with security information and constraints[3]

Fig. 7 shows a class diagram that includes three classes: *UserProfile*, *Patient*, and *Admission*. *UserProfile* class contains the information of all users who will have access to this database. *Patient* class contains the information of hospital patients, and can be accessed by all users who have *secret* or *top secret* security levels, and play *health* or *administrative* roles. *Admission* class contains the information of all hospital admissions, and can be accessed by users who play *health* or *administrative* roles, and have *secret* or *top secret* security levels. We can analyze that *diagnosis* attribute can only be accessed by users who play *health* role, and *value* attribute can only be accessed by users who play *administrative* role. There is also an association between *Patient* and *Admission* classes. A patient can be related to one or more admissions,

[3] Version 2 of OCL considers a special syntax for enumerations (EnumTypeName::EnumLiteralValue), but in this example, for the sake of readability, we consider only EnumLiteralValue.

but an admission will always be related to a patient. Several security constraints have been specified by using the previously defined constraints and stereotypes:

- The security level of each instance of *Admission* is defined by a security constraint that is specified in the model. If the value of the attribute *diagnosis* is "cancer", the security level of this admission will be *top secret*, otherwise *secret*. The security level of each instance can also depend on the value of the *value* attribute, that indicates the price of the admission service.
- The stereotype *Log* has been defined for the *Admission* class, specifying the tagged value *type=frustratedAttempts*. This stereotype specifies that the system has to record, for future audit, the situation in which an user tries to access to information of this class, and the system denies it because of lack of permissions.
- Patients could be special users of the system. In this case, it could be possible that patients access to their own information as patients (for instance, for querying their personal data). This constraint is specified by using the *Exception* stereotype in the *patient* class.
- For confidentiality reasons, we could deny access to admission information to all doctors whose date of contract with the hospital is later than the date of admission of a patient. This is specified by an exception in the *Admission* class.
- Moreover, we could also deny access to admission information to users whose working area is different than the area of a particular admission instance. This is specified by another exception in *Admission* class.

We can notice that, using this extension, it is possible to specify a wide range of confidentiality constraints in the conceptual model of a database. All these security constraints are transformed along the stages of the methodology we have mentioned before, and finally they are implemented with OLS.

Additionally, an Add-in of Rational Rose has been developed to support most of the elements we have defined in this paper. This CASE tool can be used in the secure database conceptual modeling; with the advantage of the fact that Rational Rose is one of the most used CASE tools in the software development process with UML.

For space restrictions, we refer readers to [12], to analyze how these constraints are transformed in the methodology and implemented in OLS, and to see an overview of the CASE tool.

5. Conclusions and Future Work

In this paper, we have presented an extension of UML and OCL that allows us to represent sensitivity information and constraints in the database conceptual model. This extension contains the necessary stereotypes, tagged values and constraints for integrating security in the first stages of secure database development, and according to MDA ideas, it makes it easier the task of maintaining the security information and constraints along all the process, making it possible to compile constraints and, automatically or semi-automatically generating code that implements security control in a secure DBMS. One of the most important advantages of this approach is that it uses UML, a widely-accepted object-oriented modeling language, which saves developers from learning a new model and its corresponding notations for specific models.

We are currently working on extending the functionality of the Add-in of Rational Rose in order to automatically obtain the multilevel relational model, once the secure conceptual model has been developed, and automatically compile the security constraints and obtain code to OLS. Furthermore, in this extension, we are also considering new stereotypes and constraints regarding other security problems, such as integrity and availability.

Acknowledgements

This research is part of the CALIPO (TIC2003-07804-C05-03) and RETISTIC (TIC2002-12487-E) project, supported by the Dirección General de Investigación of the Ministerio de Ciencia y Tecnología. We would like to thank Luis Reynoso for his valuable comments in the paper review process.

References

1. Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language, User Guide*. 1999, Redwood city, CA: Addison-Wesley.
2. Charpentier, R. and Salois, M. *Security Modelling for C2IS in UML/OCL*. in *8th ICCRTS*. 2003. Washington DC.
3. Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., *Non-functional requirements in software engineering*. 2000, Boston/Dordrecht/London: Kluwer Academic Publishers.
4. Conallen, J., *Building Web Applications with UML*. Object Technology Series. 2000: Addison-Wesley.
5. Cook, S., Kleppe, A., Mitchell, R., Rumpe, B., Warmer, J., and Wills, A., *The Amsterdam Manifesto on OCL*, in *Object Modeling with the OCL*, Clark, T. and Warmer, J., Editors. 2002, Springer. p. 115-149.
6. Cota, S., *For Certain Eyes Only*. DB2 Magazine, 2004. 9(1): p. 40-45.
7. Database, D.U., *DB2 UDB for Z/OS v.8*. 2004.
8. Demuth, B. and Hussmann, H. *Using UML/OCL Constraints for Relational Database Design*. in *The Unified Modeling Language*. 1999. Fort Collins, CO, USA: Springer, LNCS.
9. Devanbu, P. and Stubblebine, S., *Software engineering for security: a roadmap*, in *The Future of Software Engineering*, Finkelstein, A., Editor. 2000, ACM Press. p. 227-239.
10. Dhillon, G. and Backhouse, J., *Information system security management in the new millennium*. Communications of the ACM, 2000. **43**(7): p. 125-128.
11. Fernandez, E.B. and Pan, R.Y. *A pattern language for security models*. in *8th Conference on Patterns Languages of Programs (PLOP 2001)*. 2001. Illinois, USA.
12. Fernández-Medina, E. and Piattini, M., *Designing Secure Database for OLS*, in *Database and Expert Systems Applications: 14th International Conference (DEXA 2003)*, Marik, V., Retschitzegger, W., and Stepankova, O., Editors. 2003, Springer. LNCS 2736: Prague, Czech Republic. p. 886-895.
13. Ferrari, E. and Thuraisingham, B., *Secure Database Systems*, in *Advanced Databases: Technology Design*, Piattini, M. and Díaz, O., Editors. 2000, Artech House: London.
14. Flake, S. and Mueller, W., *An OCL Extension for Real-Time Constraints*, in *Object Modeling with the OCL*, Clark, T. and Warmer, J., Editors. 2002, Springer. p. 150-171.
15. Gogolla, M. and Henderson-Sellers, B. *Analysis of UML Stereotypes within the UML Metamodel*. in *5th International Conference on the Unified Modeling Language - The Language and its Applications*. 2002. Dresden, Germany: Springer, LNCS 2460.

16. Hall, A. and Chapman, R., *Correctness by Construction: Developing a Commercial Secure System.* IEEE Software, 2002. 19(1): p. 18-25.
17. Hamie, A., Mitchell, R., and Howse, J., *Time-Based Constraints in the Object Constraint Language.* 1999.
18. Jürjens, J., *UMLsec: Extending UML for secure systems development,* in *UML 2002 - The Unified Modeling Language, Model engineering, concepts and tools,* Jézéquel, J., Hussmann, H., and Cook, S., Eds. 2002, Springer. LNCS 2460.: Dresden, Germany. p. 412-425.
19. Kleppe, A. and Warmer, J., *Extending OCL to Include Actions,* in *UML 2002,* Evans, A., Kent, S., and Selic, B., Editors. 2000, Springer. LNCS 2460. p. 440-450.
20. Kleppe, A. and Warmer, J., *The Semantics of the OCL Action Clause,* in *Object Modeling with the OCL,* Clark, T. and Warmer, J., Editors. 2002, Springer. p. 213-227.
21. Kleppe, A., Warmer, J., and Bast, W., *MDA Explained; The Model Driven Architecture: Practice and Promise.* 2003: Addison-Wesley.
22. Levinger, J., *Oracle label security. Administrator's guide. Release 2 (9.2).* 2002: http://www.csis.gvsu.edu/GeneralInfo/Oracle/network.920/a96578.pdf.
23. Marks, D., Sell, P., and Thuraisingham, B., *MOMT: A multi-level object modeling technique for designing secure database applications.* Journal of Object-Oriented Programming, 1996. 9(4): p. 22-29.
24. Nunes, I. *An OCL Extension for Low-coupling Preserving Contracts.* in *The Unified Modeling Language, Modeling Languages and Applications.* 2003. San Francisco, CA, USA: Springer, LNCS 2863.
25. Piattini, M. and Fernández-Medina, E. *Specification of Security Constraint in UML.* in *35th Annual 2001 IEEE International Carnahan Conference on Security Technology (ICCST 2001).* 2001. London (Great Britain).
26. Samarati, P. and De Capitani di Vimercati, S., *Access control: Policies, models, and mechanisms,* in *Foundations of Security Analysis and Design,* Focardi, R. and Gorrieri, R., Editors. 2000, Springer: Bertinoro, Italy. p. 137-196.
27. Smith, G.W., *Modeling security-relevant data semantics.* IEEE Transactions on Software Engineering, 1991. 17(11): p. 1195-1203.
28. Warmer, J. and Kleppe, A., *The object constraint language.* 1998, Massachusetts: Addison-Wesley.
29. Warmer, J. and Kleppe, A., *The Object Constraint Language Second Edition. Getting Your Models Ready for MDA.* 2003: Addison Wesley.
30. Ziemann, P. and Gogolla, M. *OCL Extended with Temporal Logic.* in *Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference.* 2003. Akademgorodok, Novosibirsk, Russia: Springer LNCS.

Test Driven Development of UML Models with SMART Modeling System

Susumu Hayashi[1], Pan YiBing[1], Masami Sato[1], Kenji Mori[1], Sul Sejeon[1], and Shusuke Haruna[2]

[1] Kobe University, Nada, Kobe, Japan,
{shayashi,pybing,satoman,moriken,sejeon}@cs33.scitec.kobe-u.ac.jp
[2] Matsushita Electric Industrial Co., Ltd., Kadoma, Osaka, Japan,
haruna@isl.mei.co.jp

Abstract. We are developing a methodology for Test-Driven Development of Models (TDDM) based on an experimental UML 2.0 modeling tool SMART. Our experience shows that TDDM is quite useful for agile model developments. SMART provides guidance on how to build models based on compiler errors of testcases, something similar to what Quick Fix of Eclipse does. It also provides such guidance from failures of testcases, which seems difficult in the case of TDD of programs.

1 Introduction

The integration of Agile methods and Modeling is attracting considerable attention [1,2,5,6,11]. Test-Driven Development (TDD) is one of the central notions of Agile developments. In this paper, we present a method of TDD of Models with with an experimental UML 2.0 modeling tool called SMART.

The most important feature that distinguishes our method from other agile methods is the full scale realization of tool support of test-driven *guidance*. By "guidance", we mean "a suggestion of the next steps to be achieved and/or an aid to achieve them."

In the original TDD, compiler error messages were utilized as suggestions to inform us which classes and methods should be defined next [3]. Quick Fix of Eclipse [15,7] and certain other tools aid us in writing them by automatically generating stubs.

Thanks to UML architectures, we were able to take the idea of guidance forward even more than the original TDD for codes. Quick Fix guides which and how program elements such as classes and methods should be introduced. However, Quick Fix guidance is restricted to *syntactical* aspects, since it is made based on compiler errors (see [7,15]). Failures of testcases can provide us with behavioral information how to fix codes. SMART system supports even TDD of behavioral aspects based on failures of testcases and other execution information. Quick Fix generates stubs but our method can generate fakes as well. (A stub does nothing. It exists only for compilation. A fake returns a fixed correct value for a particular testcase. See p. 169 of [2].)

T. Baar et al. (Eds.): UML 2004, LNCS 3273, pp. 395–409, 2004.