UPGRADE
The European Journal for the Informatics Professional
http://www.upgrade-cepis.org
Vol. V, No. 2, April 2004

UPGRADE is the European Journal for the Informatics Professional, published bimonthly at <http://www.upgrade-cepis.org/>

Publisher
UPGRADE is published on behalf of CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>) by NOVÁTICA <http://www.ati.es/novatica/>, journal of the Spanish CEPIS society ATI (Asociación de Técnicos de Informática <http://www.ati.es/>).
UPGRADE is also published in Spanish (full issue printed, some articles online) by NOVÁTICA, and in Italian (abstracts and some articles online) by the Italian CEPIS society ALSI <http://www.alsi.it> and the Italian IT portal Tecnoteca <http://www.tecnoteca.it/>.
UPGRADE was created in October 2000 by CEPIS and was first published by NOVÁTICA and INFORMATIK/INFORMATIQUE, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <http://www.svifsi.ch/>).

E-mail addresses for editorial correspondence: <rfcalvo@ati.es>, <nicolet@acm.org> or <rcarniel@dgt.uniud.it>

E-mail address for advertising correspondence: <novatica@ati.es>

Upgrade **Newslist** available at <http://www.upgrade-cepis.org/pages/editinfo.html#newslist>

ISSN 1684-5285

Next issue (June 2004):
"Digital Signature"
(The full schedule of **UP**GRADE is available at our website)

2 From the Editors' Desk
The **UP**GRADE European Network: *N przywitanie* / Welcome!

*The members of the Editorial Team of **UP**GRADE describe the aims and scope of the network of journals of CEPIS member societies, whose contents will enrich ours and offer a broader European view of ICT to our readership.*

UML and Model Engineering

Guest Editors: **Jesús García-Molina, Ana Moreira, and Gustavo Rossi**

Joint issue with NOVÁTICA*

3 Presentation
UML: The Standard Object Modelling Language – *Jesús García-Molina, Ana Moreira, and Gustavo Rossi*

The guest editors introduce the monograph, that includes a series of papers that reflect the state of the art of UML (Unified Modeling Language). These papers illustrate different aspects of UML, ranging from use cases to UML formalization, meta-modelling, profile definition, model quality, model engineering and MDA (Model Driven Architecture.)

6 An Introduction to UML Profiles – *Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno*

This paper describes a set of steps to create a profile and argue the importance of profiles in MDA.

14 Aspect-Oriented Design with Theme/UML – *Siobhán Clarke*

The author describes her approach "Theme" to extending the UML in order to support the modularisation of a designer's concerns, including crosscutting ones.

21 In Search of a Basic Principle for Model Driven Engineering – *Jean Bézivin*

This article offers an interesting look at the essential features of this new software development paradigm.

25 The Object Constraint Language for UML 2.0 – Overview and Assessment – *Heinrich Hussmann and Steffen Zschaler*

This paper, authored by members of the OCL 2.0 team, gives an overview of the new aspects of the second version of this language and also provides a critical discussion of a few selected aspects of it.

29 Developing Security-Critical Applications with UMLsec. A Short Walk-Through – *Jan Jürjens*

The problems of creating high-quality critical systems is analysed in this paper, that shows how using UML modelling can help solve them and presents a tool to support the proposed approach.

36 On the Nature of Use Case-Actor Relationships – *Gonzalo Génova-Fuster and Juan Llorens-Morillo*

In this paper some issues are addressed that regard the relationships in which use cases and actors may take part, presently defined in UML as associations.

43 Metrics for UML Models – *Marcela Genero, Mario Piattini-Velthuis, José-Antonio Cruz-Lemus, and Luis Reynoso*

This paper offer a vision of the state of the art of metrics for measuring quality of some basic UML diagrams (such as class, state and use case diagrams) and OCL expressions.

49 Using Refactoring and Unification Rules to Assist Framework Evolution – *Mariela I. Cortés, Marcus Fontoura, and Carlos J.P. de Lucena*

In their paper the authors use UML-F, a UML designed for describing frameworks, to present two techniques aimed at facilitating framework maintenance and evolution.

Mosaic

UPGRADE European Network

From "Pro Dialog" (Poland):

56 Parallel Programming Support System for Transputers – Educational Software – *Mikolaj Szczepanski and Rafal Walkowiak*

The paper presents a method for integrating applications data, aimed at data aggregation and transfer in software applications when integration of those applications has to be fast and should be done with minimum source code modifications.

News Sheet

61 ENISA: The European Network and Information Security Agency created

61 News from EUCIP and ECDL

* This monograph will be also published in Spanish (full issue printed; summary, abstracts and some articles online) by NOVÁTICA, journal of the Spanish CEPIS society ATI (Asociación de Técnicos de Informática) at <http://www.ati.es/novatica/>, and in Italian (online edition only, containing summary abstracts and some articles) by the Italian CEPIS society ALSI and the Italian IT portal Tecnoteca at <http://www.tecnoteca.it>.

An Introduction to UML Profiles

Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno

UML (Unified Modeling Language) has become one of the most widely used standards for specifying and documenting information systems. However, the fact that UML is a general purpose notation may limit its suitability for modelling some particular domains, for which specialized languages may be more appropriate. UML provides a set of extension mechanisms to address this issue, which allow the customisation and extension of its own syntax and semantics in order to adapt to certain application domains. In this paper we examine the extension mechanisms which are used to define UML Profiles. We also discuss the usefulness and relevance of UML Profiles in the context of Model Driven Architecture (MDA®).

Keywords: MDA, UML Modelling, UML Profiles.

1 Introduction

The increasing complexity of information systems is challenging the way software architects and engineers work. After initially being concerned more about the structure and quality of programming code, software engineers are now focusing their attention on the modelling aspects of the system development process. Models provide abstractions of systems which help deal with larger and more complex applications in simpler ways, regardless of how they are implemented and distributed and whichever the final execution platform or technology used.

A model is a description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax) and meaning (semantics), which is suitable for automated interpretation by a computer [1].

The OMG (Object Management Group) defines several modelling languages, among which UML (Unified Modeling Language) [2] is probably the one most widely accepted and used. UML is a visual language for specifying, constructing and documenting the artefacts of systems. It is a general purpose modelling language that can be used with all major object and component methods and can be applied to all application domains (e.g. health, finance, telecom, aerospace) and implementation platforms (e.g., CORBA – Common Object Request Broker Architecture –, J2EE – Java 2 Enterprise Edition –, .NET).

There are situations, however, in which a language that is so general and of such a broad scope may not be appropriate for modelling applications of some specific domains. This is the case, for instance, when the syntax or semantics of the UML elements cannot express specific concepts of particular systems, or when we want to restrict or customize some of the UML elements which are usually too abundant and too general.

OMG defines two possible approaches for defining domain-specific languages. The first one is based on the definition of a new language, an alternative to UML, using the mechanisms provided by OMG for defining object-based visual languages (i.e. the same mechanisms that have been used for defining UML and its metamodel). In this way, the syntax and semantics of the elements of the new language are defined to fit the specific characteristics of the domain. The second alternative is based on UML specialisation, in which some of the language's elements are specialised, imposing new restrictions on them, while respecting the UML metamodel and leaving the original semantics of the UML elements unchanged (i.e. the properties of the UML classes, associations, attributes etc. will remain the

Lidia Fuentes-Fernández received her MSc degree in Computer Science from the *Universidad de Málaga*, Spain, in 1992 and her doctorate in 1998 from the same university. She has been an Associate Professor in the Department of Computer Science of the *Universidad de Málaga* since 1993. Her research interests centre around Component-Based Software Development, frameworks, aspect orientation, software agents and multimedia applications. She has also participated in several international conferences as a speaker, reviewer, and chairperson. Her most significant publications are to be found in international journals published by ACM and IEEE, such as IEEE Internet Computing, IEEE Transactions on Software Engineering, ACM Computing Surveys or Software Practice and Experience. She has actively participated in several research projects and she has also done consulting work for telecommunication companies such as Telefónica and Retevisión, Spain. <lff@lcc.uma.es>

Antonio Vallecillo-Moreno holds BSc and MSc degrees in Mathematics, and a PhD degree in Computer Science. Most of his professional experience has been in the computer industry, where he has worked for many years in various international companies both in Spain and in the UK. He now works at the *Universidad de Málaga*, Spain, where he is currently an associate professor, and he has also been Head of the University IT Services. His research interests include component-based software development, open distributed processing, and the industrial use of formal methods. He is the *Universidad de Málaga* representative for ISO, the OMG, and AENOR (the Spanish National Body for Standardization), and a member of several professional organizations, including the ACM and the IEEE. <av@lcc.uma.es>

same; new constraints will simply be added to their original definitions and relationships). Syntactic sugar can also be used in a Profile in the form of icons and symbols for newly defined elements.

The first approach is that followed by languages such as CWM (Common Warehouse Metamodel) [3], as the semantics of some of the language constructs do not match the semantics of the corresponding UML model elements. These new languages are defined using the MOF (Meta Object Facility), a language specifically designed for defining object-based modelling languages. As we will discuss later, UML itself is defined using the MOF.

In order to support the second alternative, UML provides a set of extension mechanisms (stereotypes, tagged values, and constraints) for specializing its elements, allowing customized extensions of UML for particular application domains. These customisations are sets of UML extensions grouped into *UML Profiles*.

Each alternative has its advantages and disadvantages. Defining a tailor-made language will produce a notation that will perfectly match the concepts and nature of the specific application domain. However, as the new language does not respect UML semantics, it will not allow the use of commercial UML tools for drawing diagrams, generating code, reverse engineering, and so forth. Conversely, UML Profiles (which are amenable to be handled by commercial UML tools) may not provide such an elegant and perfectly fitting notation as may be required for those systems. It is not therefore always easy to decide when to create a new language and when to define a set of extensions to the standard UML metamodel by grouping them into a UML Profile. In our experience, unless there is a real need to deviate from the UML metamodel, the benefits of using UML Profiles undoubtedly outweigh their limitations.

In this paper we will introduce the extension mechanisms used in the definition of a UML Profile. We will also discuss the usefulness and importance of UML Profiles in the context of Model Driven Architecture (MDA®) [4]. MDA is an OMG initiative that provides an approach to system development based on model definition and transformations. It is model driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification. As we will see, UML Profiles play an important role in MDA.

UML Profiles were originally defined in version 1 of UML, although their applicability and widespread use by the software community was limited because they lacked either an unambiguous definition or precise utilization guidelines [5]. The new version of UML (2.0) addresses these issues, providing substantial improvements over the UML Profiles of UML version 1. For instance, among the improvements introduced, there is a better conceptual alignment with the MOF, and with the UML metamodel (UML Profiles can extend not only UML but also any other MOF-defined language, or even other UML Profiles). Also the UML model elements to be specialized are now more clearly and conveniently specified, and the profile notation has been enhanced.

This is an introductory paper and is intended only as a brief presentation of UML Profiles to show something of their potential. We refer interested readers to the UML 2.0 Infrastructure Specification (OMG document ptc/03-12-01) for more technical information and further details on the subject.

This paper is structured in six sections, this introduction being the first. Section 2 presents the four-layered conceptual architecture defined by OMG for modelling systems and defining modelling notations such as UML or CWM. Section 3 introduces UML Profiles, as defined in UML 2.0., and Section 4 provides some brief guidelines for the definition and usage of UML Profiles. The role of UML Profiles in MDA is discussed in Section 5. Finally, Section 6 draws some conclusions and outlines some of the still unresolved limitations of UML Profiles.

2 OMG's Metamodel Architecture

In the previous Section, a *model* was defined as a description of (part of) a system written in a well-defined language. The problem is how to define such a *well-defined* language.

The solution to this problem is well known in the case of textual languages, whose grammar is described using the Backus Naur Form (BNF) notation. However, this notation does not work for defining graphical languages, for which a different mechanism is needed. This mechanism is called *metamodelling*.

As a general rule we can suppose that a model describes the elements and types of elements that might exist in a system. For example, if we define "Person" as a class in a model, we can use instances of that class in our system (such as "John" or "Mary"). Following that principle, if the system we want to model is a UML system, then the elements comprising it will be "Class", "Association", "Package", etc. But who defines those elements, and how are they defined?

The OMG defines a four-layered architecture that separates the different conceptual levels making up a model: the instances, the model of the system, the modelling language, and the metamodel of that language. In OMG terminology these layers are called M0, M1, M2, and M3.

- **Layer M0: Instances.** The M0 layer models the running system and its elements are the actual instances that exist in the system. These instances are, for example, customer "John" who lives at "1 Oxford St., London" and buys the copy number "123" of the book "Ulysses".
- **Layer M1: The model of the system.** The elements of the M1 layer are *models*. An example would be a UML model of a software system. The M1 layer defines the *classes* "Customer", "Address", "Purchase" and "Book", each one with its associated *attributes* ("address", "copy no.", "title", etc.). There is a strong relationship between the M0 and M1 layers. The elements of the M1 layer are *classifications* of elements of the M0 layer. Likewise, each element at the M0 layer is always an *instance* of an element at the M1 layer.
- **Layer M2: The model of the model (the *metamodel*).** The elements of layer M2 are the modelling languages. Layer M2 defines the concepts that are used to model an element of layer M1. In the case of UML, layer M2 defines "Class",

"Attribute", "Association", etc. Just as there was a close relationship between layers M0 and M1 so there is a close relationship between M1 and M2 layers. Every element at M1 is an *instance* of an M2 element, and every element at M2 *categorizes* M1 elements. The model that resides at the M2 layer is called a metamodel.

- **Layer M3: The model of M2 (the *meta-metamodel*).** Finally, layer M3 defines the concepts that can be used to define modelling languages. Thus, the concept of UML Class (that belongs to M2) can be considered as an *instance* of the corresponding element of M3, which precisely defines what a Class is and its relationships with the rest of the UML concepts (e.g., a UML Class is a *classifier*, and therefore has an associated behaviour, and a set of attributes and methods, etc.).

The modelling language defined for describing the M3 elements is called MOF (Meta-Object Facility) [6]. MOF is a language to define modelling languages, such as UML, CWM, or even MOF itself. Such languages can be considered as *instances* of MOF.

To summarise, a well-defined language (such as UML) can be described by its metamodel. What MOF provides is a language to describe metamodels. If we wanted to define a new object-based visual language other than UML we would use the MOF to describe its metamodel.

3 UML Profiles

For when we need to define a new language to model a system that either restricts the number of UML elements or adds some constraints or syntactic sugar to them while respecting the original semantics, we do not need to create a new language from scratch using the MOF. Instead, UML can easily be customized by using a set of extension mechanisms that UML itself provides. More precisely, the Profiles package included in UML 2.0 defines a set of UML artefacts that allows the specification of an MOF model to deal with the specific concepts and notation required in particular application domains (e.g., real-time, business process modelling, finance, etc.) or implementation technologies (such as .NET, J2EE, or CORBA). It should be noted that UML Profiles allow the customisation of any MOF defined (not just UML defined) metamodel. Similarly, a UML Profile can also specify another UML Profile.UML 2.0 outlines several reasons why a system developer should want to customize his metamodel:

- To have a terminology that is adapted to a particular platform or domain (for example, being able to capture EJB (Enterprise Java Beans) terminology such as "home interface", "enterprise java beans", "archive" etc.).
- To have a syntax for constructs that do not have a notation (as is the case of actions in UML).
- To have a different notation for already existing symbols, more appropriate for the target application domain (such as being able to use a picture of a computer instead of the ordinary UML node symbol to represent a computer in a network).
- To add semantics left unspecified in the metamodel (such as how to deal with priority when receiving signals in a state machine).
- To add semantics that do not exist in the metamodel (such as defining a timer, clock, or continuous time)
- To add constraints that restrict the way you can use the metamodel and its constructs (such as disallowing actions from being executable in parallel within a single transition, or forcing the existence of certain associations between model classes)
- To add information that can be used when transforming one model to another model or to code (such as defining mapping rules between a model and Java code).

A UML Profile is defined as a UML package stereotyped "profile", that can extend either a metamodel or another Profile. UML Profiles are defined in terms of three basic mechanisms: *stereotypes*, *constraints*, and *tagged values*.

Let's take a look at an example to define and illustrate all these concepts (see Figure 1). Suppose we want to add two new elements to our UML models – say, weights and colours – and we want to do so in a precise and consistent way. Furthermore, we may want to incorporate some particular properties and requirements of such elements, such as the actual colour of a coloured element, the weight of a weighed element, and a restriction that states that coloured associations can only be defined between classes of the same colour as that of the association. For the sake of simplicity, we will assume here that only classes and associations can be coloured, and that only associations can be weighed.

The WeightsAndColours profile defines these two elements:

(1) First, a *stereotype* is defined by a name and by the set of metamodel elements it can be attached to. Graphically,

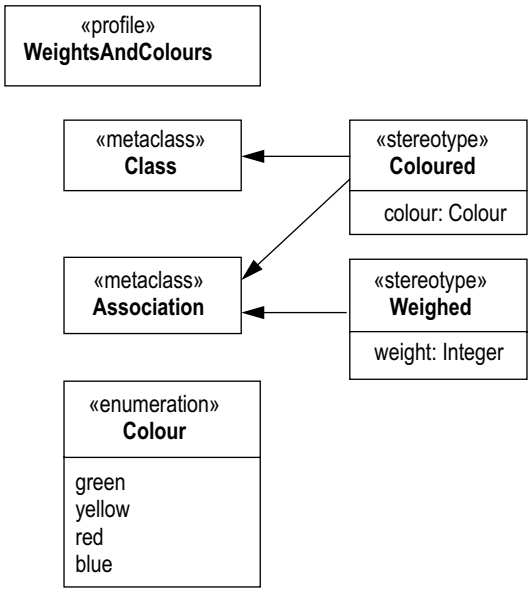

Figure 1: Example of UML Profile.

stereotypes are defined within boxes, stereotyped «stereotype». In the example, the WeightsAndColours UML Profile defines two stereotypes, Colored and Weighed, and indicates that both UML classes and associations can be coloured (i.e, stereotyped «Colored»), but only associations can have an associated weight (i.e, stereotyped «Weighed»). Metamodel elements are indicated by classes stereotyped «metaclass». The notation for an extension is an arrow pointing from a stereotype to the extended class, where the arrowhead is shown as a solid triangle. An Extension may have the same adornments as an ordinary association, but navigability arrows are never shown.

(2) *Constraints* can be associated to stereotypes, imposing restrictions on the corresponding metamodel elements. In this way a designer can define the properties of a "well-formed" model. For instance, the aforementioned restriction on the colours of the classes joined by a coloured association can be expressed by the following OCL [7] constraint:

```
context UML::InfrastructureLibrary::Core::
         Constructs::Association
  inv : self.isStereotyped("Coloured") implies
        self.connection->forAll
        (isStereotyped("Coloured")
        implies color=self.color)
```

Constraints can be expressed in any language, including natural language or the OCL (Object Constraint Language). OCL is a language, now part of UML, adopted by the OMG for expressing constraints and properties of model elements. Examples of constraints include pre- and post-conditions of operations, invariants, derivation rules for attributes and associations, the body of query operations, etc. The word "Constraint" in the name of the language comes from its first version, where only constraints could be expressed. New OCL 2.0 has evolved to a more expressive and powerful query language, whose expressiveness is similar to that of SQL.

(3) Finally, a *tagged value* is an additional meta-attribute that is attached to a metaclass of the metamodel extended by a Profile. Tagged values have a *name* and a *type*, and are associated to a specific stereotype. In the example, the stereotype «Weighed» has an associated tagged value named "weight", of type Integer, that represents the actual weight of the stereotyped association. Graphically, tagged values are specified as attributes of the class that defines the stereotype.

A UML Profile is a set of these extension mechanisms, grouped into a UML package stereotyped «profile». As mentioned earlier, note that these mechanisms allow the extension of the syntax and semantics of UML elements but must respect their original semantics, i.e., UML Profile cannot change the semantics of UML elements. However they can be very useful when customisations of UML are required for particular application domains.

Several UML Profiles currently exist and are available for public use. Some of them have even been adopted and standardized by the OMG, such as the UML Profile for CORBA and for CCM (CORBA Component Model), the UML Profile for EDOC (Enterprise Distributed Object Computing), the UML Profile for EAI (Enterprise Application Integration), and the UML Profile for Scheduling, Performance, and Time. Some other UML Profiles have already been defined, and are now in the process of being approved by the OMG, so the number of OMG Profiles is rapidly growing. One of the benefits of UML Profiles is that they can be directly reused in any application, as we shall see later.

Other UML Profiles have been defined by private organizations and software companies, and are currently being used in many application domains (hence becoming de facto standards). This is the case, for instance, of the "UML/EJB Mapping Specification", a UML Profile for EJB applications that has been defined by JCP (Java Community Process). UML Profiles for programming languages such as Java or C# are also available.

Each of these profiles defines a precise way to use UML in a particular context. For instance, the UML Profile for CORBA defines a way in which UML can be used to model CORBA interfaces and artefacts; and the UML Profile for Java defines a way of modelling Java constructs and applications using UML. Furthermore, such Profiles can be combined within the MDA context to define a chain of model transformations: from a model of the system independent from the implementation platform, to the corresponding model of the system on a CORBA platform; and then from the CORBA model to a model of its implementation using Java (which "almost" coincides with its implementation).

4 Definition of UML Profiles

In this section we present some brief guidelines for the definition and use of UML Profiles. The UML 2.0 specification [8] merely defines the concept of Profile and the elements that comprise a Profile definition. However, we consider that users may also need some hints and recommendations to help them define a UML Profile for a given platform or application domain.

The steps that we propose for defining a UML Profile are the following:

1. First of all, we need to define the set of elements that will comprise our platform or system, and the relationships between them, which can be expressed in terms of a metamodel. If we do not have such a metamodel, we can easily define it using the standard mechanisms offered by UML (classes, hierarchy relationships, associations, etc.) in the normal way, i.e., as if we did not intend to build a UML Profile for it. In the metamodel we need to include the definition of the domain entities, the relationships between them, and the constraints that govern both the structure and behaviour of these entities.
2. Once we have our domain metamodel we are ready to define the UML Profile. We will include a stereotype for each relevant element of the metamodel that we want to include in the Profile, inside the «profile» package. In order to clarify the relationship between the metamodel and the Profile, these stereotypes will be named after the corre-

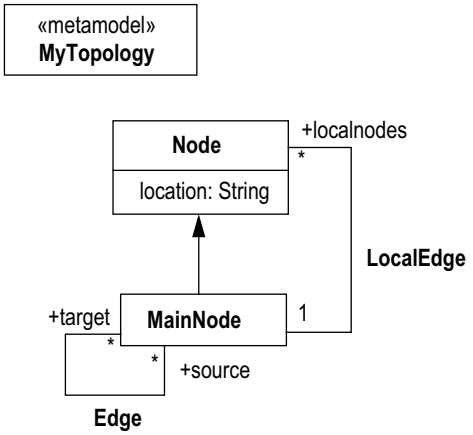

Figure 2: Metamodel Describing an Application Domain.

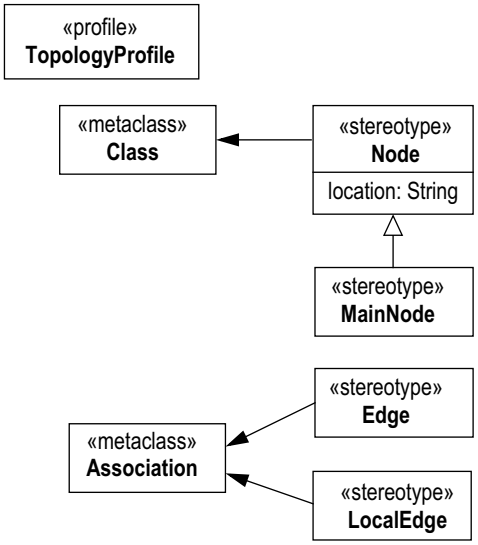

Figure 3: UML Profile Described as a UML Package.

sponding elements of the metamodel. In fact any element that we might need to define the metamodel can later be tagged with a stereotype.

3. Note that we will only represent with a stereotype the elements of the UML metamodel we are extending. Examples of these elements are classes, associations, attributes, operations, transitions, packages, etc. In this way, each stereotype will be applied to the UML metaclass that we have used in the domain metamodel for defining a concept or a relationship.
4. We should define the attributes that appear in the metamodel as tagged values, and include the corresponding types and initial values.
5. Profile constraints are taken from the domain restrictions. For example, the association multiplicities that appear in the domain metamodel or the business rules of the application are used to define the corresponding OCL constraints.

Let us illustrate the use of these guidelines with another example. Suppose we need to model the connections between the elements of an information system with a star topology, in which nodes connect to central nodes which can then be connected between each other. In such a domain we would define nodes (represented by class Node) connected by links that can be local nodes (LocalEdge) if they connect nodes from the same star with its central node, or remote nodes (MainNode) if they connect central nodes between each other. Each node is identified by its position (location). We impose the constraint that every node of the same star must share the same geographic location. In Figure 2 the metamodel describing this application domain is depicted.

As part of this metamodel, we also need to specify the set of constraints that govern its structure – i.e., its "well-formedness" rules. In this case, these constraints can be described in OCL as follows:

```
context MyTopology::MainNode
  inv:self.localnodes ->forAll (n : Node |
       n.location = self.location)
  inv:self.target ->forAll (n : MainNode |
       n.location <> self.location)
```

The UML Profile that represents this metamodel will be described as a UML package, with the stereotype «profile» (see Figure 3.)

This Profile defines four different stereotypes that correspond to the classes and associations defined in the original metamodel, as well as the UML metaclasses to which these stereotypes can be applied. These metaclasses are part of the metamodel to be extended, in this case the UML metamodel.

The stereotype Node also has a tagged value (location) of type String.

In addition to stereotypes and tagged values, we need to specify the constraints of the Profile. Coming back to our example, the restrictions of the metamodel result in the following constraint specification.

```
context  UML::InfrastructureLibrary::Core::
         Constructs::Class
  inv :  - Nodes should be locally
         connected to exactly one MainNode
         self.isStereotyped("Node")
         implies
             self.connection->select
             (isStereotyped("LocalEdge"))->
             size = 1 and
             self.connection->select
             (isStereotyped("Edge"))-> isEmpty
context  UML::Infrastructure
         Library::Core::Constructs::
         Association
  inv :  self.isStereotyped("LocalEdge")
         implies
             self.connection->select
                   (participant.isStereotyped
                   ("Node") or
                   participant.isStereotyped
                   ("MainNode") ) ->
               forAll(n1, n2 | n1.location =
               n2.location)
```

```
inv :  self.isStereotyped("LocalEdge")
       implies
       self.connection-> exists
       (participant.isStereotyped
       ("MainNode") and
       multiplicity.min=1 and
       multiplicity.max=1)
inv :  self.isStereotyped("Edge")
       implies
       self.connection->select(participant.
       isStereotyped("Node"))->isEmpty and
       self.connection->select
       (participant.isStereotyped
       ("MainNode") ) ->
              forAll(n1, n2 | n1.location <>
              n2.location)
```

Note that the constraint contexts are the UML elements that we are customizing for our particular model. Customisation is therefore achieved by adding semantic constraints to them. One of the most important advantages of using UML Profiles is that these constraints can be checked in the final system model for conformity with a given Profile. This means that UML Profiles always describe a "well-formed" model for an application domain, and ensures that a given model will always comply with the syntactic or semantic constraints defined by whichever UML Profile is used.

Once we have defined a UML Profile, we use a dependency relationship (stereotyped as «apply») to show we are using this Profile in a specific application. For example, the diagram in Figure 4 shows an application that makes use of the UML Profiles we have defined before.

This application can therefore describe diagrams like the following one which shows two classes linked by an association. Both classes are stereotyped as nodes of a star topology. One of them is also a central node. Notice how we specify the value of tagged values associated to stereotyped elements, by including notes that show the corresponding stereotype, the name of the tagged value, and the value assigned to it.

We can also show tagged values related to different stereotypes in the same UML note as the entity CentralOffice illustrates in Figure 5.

5 MDA and UML Profiles

MDA (Model Driven Architecture) is a recent initiative from the OMG that supports the definition of models as first class elements for the design and implementation of systems. According to the MDA approach, the most important activities

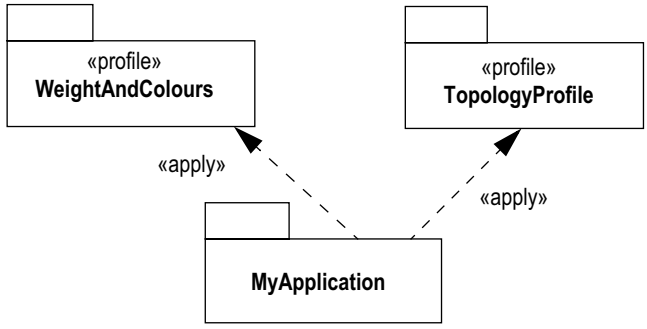

Figure 4: Application Making Use of UML Profiles.

are now modelling the different aspects of a system and then defining transformations from one model to another one in a way that allows them to be automated. We will therefore focus on model definition, leaving implementation details until the end, which makes these models more portable, more adaptable to new technologies (e.g. .NET, J2EE or Web Services) and more interoperable with other systems regardless of the technology they use.

MDA defines three conceptual levels. At the first level, system requirements are modelled in a Computation Independent Model (CIM) that defines the system within an operating environment. At the next level we find the Platform Independent Model (PIM). A PIM describes the system functionality, but without considering details about where and how this system is going to be implemented (e.g. a PIM can be independent from system distribution and the supporting object platform, such as CORBA, J2EE, .NET, etc.). The aim is then to transform a PIM into a target platform dependent model known as a PSM (Platform Specific Model). In this way we obtain a high degree of independency between the description of functionality and the implementation details of the target platform.

The most important advantage of this approach is that it allows software engineers to define automatic transformations from PIMs to PSMs. By inputting the system PIM, a description of the PSM to be used to implement the system, and a set of transformation rules, we will be able to implement a system in the most automated way possible.

UML Profiles can play a particularly important role in describing the platform model and the transformation rules between models. If we use UML Profiles to specify the model of a specific platform, this will guarantee that the derived

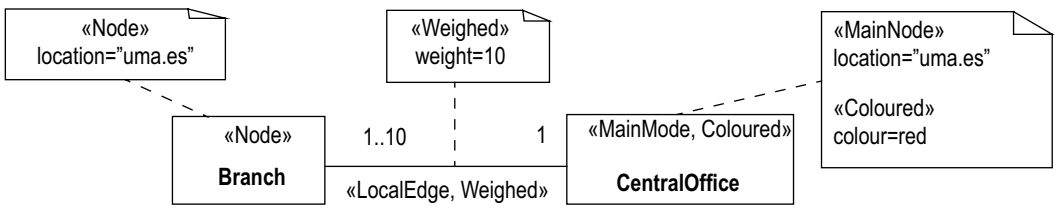

Figure 5: Tagged Values Related to Different stereotypes in the Same UML Note.

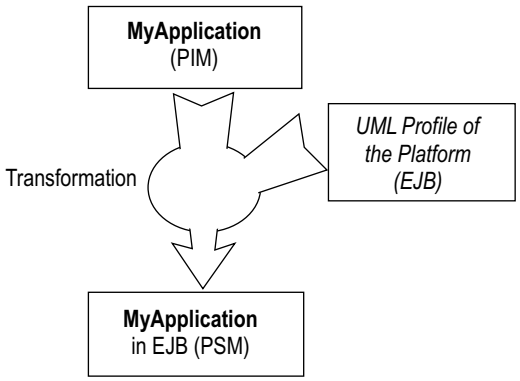

Figure 6: Transformation of a System According to an UML Profile for EJB.

models will be consistent with UML. In fact, the key to a successful application of MDA is to use standards whenever possible (standard models and standard UML Profiles for implementation languages or platforms). As we mentioned earlier, UML Profiles for some well known component platforms are currently available, such as CORBA, CCM, EJB, etc.

The mechanisms provided by UML Profiles are very well suited to describing models for any implementation platform. We need to define the mapping between each element of the PIM, and the stereotypes, constraints and tagged values that make up the platform Profile.

The idea is to use the stereotypes of a platform Profile to "mark" the elements of a PIM and produce the corresponding PSM, already expressed in terms of the elements appearing in the target platform. A mark represents a concept in the PSM, and is applied to an element of the PIM to indicate how it must be transformed into the target PSM. Marks can also be used to specify quality of service or any extra functional requirements applicable to the final implementation. For example, some elements of the PIM could be marked as persistent or under special security conditions.

The set of marks and the transformation rules governing the use of marks must be specified in a structured way and normally they are provided as part of the UML Profile of the target platform. If the UML Profile of a platform includes the specification of operations, then the transformation rules may be specified using operations.

Coming back to our example, according to MDA principles our system MyApplication, which so far has no details about the target implementation platform, should now be transformed into one of the available platforms. In this particular case, shown in Figure 6, we are going to transform this system according to the UML Profile for EJB [9]. To do this, we need to decide for each entity of our model whether it is going to be transformed into a Bean or into a simple Java class. For example we may specify that classes stereotyped as «Node» will be Java classes, but those stereotyped as «MainNode» will be transformed into a «EJBEntityBean» component. According to the UML Profile of the EJB model we must include the definition of the «EJBRemoteInterface», the «EJBEntityHomeInterface» and the «EJBImplementation» classes within the «EJBEntityBean» component. On the other hand, our application requires that the attributes of this component must be made persistent using the EJB container model. This can be done simply by marking the attributes that we want to make persistent, in this case the location attribute, with the «EJBCmpField» stereotype, by binding the value Container to the EJBPersistenceType tag indicating that the persistence property will be managed by the EJB container.

6 Conclusions

In this paper we have presented UML Profiles as a very suitable way to extend UML in order to customize it for specific platforms and application domains. We have also illustrated the importance of UML Profiles in the development of Model Driven Applications (MDA) which place system modelling, not coding, at the cornerstone of system development. The role of UML Profiles is crucial in MDA model definition and transformation.

UML 1.5 is currently the version which is standardized and supported by commercial tools. However, the new UML version 2.0 is already defined and in the process of being approved as an OMG standard. This new version is far more complete than the previous one, offering a great many advantages and successfully addressing many of the limitations of version 1.5. For example, this new version has a better metamodel structure, a more precise semantic definition of many of the UML concepts, better alignment with the MOF metamodel and with the rest of the OMG family of languages, extensions to manage software architectures and diagram exchange, etc.

Current tools allow the definition and usage of UML Profiles, but only at a diagrammatic level, i.e., only graphically. This means that verification of constraints associated to stereotypes is not yet supported, and therefore well-formed rules can be neither checked nor enforced. The user can therefore never be sure whether or not the system being specified using a given Profile is conformant with Profile rules. It will still be some time before new tools appear that will support the definition of UML 2.0 Profiles, manage the definition of new stereotypes and tags properly and allow constraints defined by the Profiles to be checked. Once such tools are available UML Profiles will be recognised as good practice for systems specification and implementation.

Acknowledgements

CORBA™, CWM™, MDA™, UML™, and MOF™ are either registered trademarks or trademarks of the Object Management Group, Inc. in the United States and/or other countries.

References

[1] A. Kleppe, J. Warmer and W. Bast. "MDA Explained: The Model Driven Architecture™ : Practice and Promise", Addisson-Wesley, 2003.

[2] ISO/IEC. Unified Modeling Language (UML). Version 1.5. International Standard ISO/IEC 19501.

[3] Object Management Group. Common Warehouse Metamodel (CWM) Specification. OMG document, ad/2001-02-01. 2001.

[4] Object Management Group. MDA Guide Version 1.0.1. OMG document, omg/2003-06-01, 2003.

[5] L. Fuentes, A. Vallecillo and J.M. Troya. Using UML Profiles for Documenting Web-Based Application Frameworks. Annals of Software Engineering, 13: pp. 249–264, 2002.

[6] Object Management Group. Meta Object Facility (MOF) Specification. OMG document: formal/2002-04-03. 2003.

[7] Object Management Group. Object Constraint Language Specification. OMG document ad/02-05-09, 2002.

[8] Object Management Group. UML 2.0 Infrastructure Specification, OMG document ptc/03-09-15, 2003.

[9] Sun Java Community Process JSR-26. Public Draft. UML Profile for EJB, 2001.