Jasmine Farhad
Dept of Computer Science
University College London

13-Dec-02

The UML Extension Mechanisms

Introduction

There is an important need for organisations to evolve in today's market. This has led to a knowledge revolution, which enables organisations to be agile and responsive by strategically capturing, communicating and leveraging intellectual assets. Object orientation and component-based development are the backbone of this revolution and it is the Unified Modelling Language (UML) that brings these together into an essential foundation for evolution. But it is the UML extension mechanisms that take this a step further and make it possible for an organisation to truly evolve in an agile and responsive fashion, by strategically capturing, communicating and leveraging intellectual assets.

The UML is a general purpose, tool supported, and standardised modelling language that is used in order to specify, visualise, construct and document all the elements of a wide range of system intensive processes. It promotes a use case driven, architecture centric, iterative and incremental process, which is object oriented and component-based. The UML is broadly applicable to different types of systems, domains, methods and processes, which is why it is such a popular and broadly used language.

However, even though the UML is very well-defined, there might be situations in which you might find yourself wanting to bend or extend the language in some controlled way to tailor it to your specific problem domain in order to simplify the communication of your objective. This is where the UML extension mechanisms come in.

There exist four common mechanisms that can be used consistently throughout the language - namely specifications, common divisions, adornments, and extensibility mechanisms – which we are going to deal with here; however, the main focus of this work will be on the latter of these four, i.e. the extensibility mechanisms.

Specifications

The first extension mechanism that was mentioned earlier is called specifications. This is a very easy term to grasp as we all know what a specification means in our everyday language. In the UML it is just as simple.

By using a specification, we are basically specifying something in a bit more detail so that the role and meaning of the term being specified is presented to us in a more clear and concise manner. For example, we can give a class a rich specification by defining a full set of attributes, operations, full signatures, and behaviours. We will then have a clearer notion of what the capabilities and limitations of that class are. Specifications can be included in the class, or specified separately.

Common Divisions

This is the second extension mechanism that is provided to us by the UML. Common divisions are used in order to distinguish between two things that might appear to be quite similar, or closely related to one another. There exist two main common divisions: abstraction vs. manifestation and interface vs. implementation.

In the former, we mainly talk about the distinction between a class and an object, where the class is an abstraction and the object is a clear manifestation of that class. Most UML building blocks have this kind of class/object distinction, e.g. use case, use case instance etc.

In the second common division – interface vs. implementation – we say that an interface declares some kind of contract, or agreement, whereas an implementation represents one concrete realisation of that contract. The implementation is then responsible for carrying out the interface.

Adornments

Adornments are textual or graphical items, which can be added to the basic notation of a UML building block in order to visualise some details from that element's specification. For example, let us consider association, which in its most simple notation consists of one single line. Now, this can be adorned with some additional details, such as the role and the multiplicity of each end (see fig1).

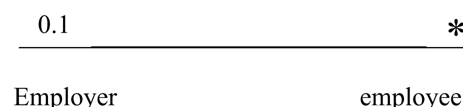

Fig 1. Association

One of the most important kinds of adornments is a *note*. This is a graphical symbol, which is used for adding some comments or constraints to an element (or a collection of elements) to help clarify the models that are being created. We may use notes in order to attach some additional information to our model, such as an explanation, a requirement, or just simply an o observation (see fig2).

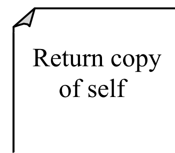

Fig 2. Note

It is worth mentioning that a note carries no semantic impact, i.e. the content of a note does not in any way change the meaning or significance of the model to which it is attached.

Extensibility Mechanisms

The extensibility mechanisms allow you to customize and extend the UML by adding new building blocks, creating new properties, and specifying new semantics in order to make the language suitable for your specific problem domain. There are three common extensibility mechanisms that are defined by the UML: stereotypes, tagged values, and constraints.

Stereotypes

Stereotypes allow you to extend the vocabulary of the UML so that you can create new model elements, derived from existing ones, but that have specific properties that are suitable for your problem domain. They are used for classifying or marking the UML building blocks in order to introduce new building blocks that speak the language of your domain and that look like primitive, or basic, model elements.

For example, when modelling a network you might need to have symbols for representing routers and hubs. By using stereotyped nodes you can make these things appear as primitive building blocks.

As another example, let us consider exception classes in Java or C++, which you might sometimes have to model. Ideally you would only want to allow them to be thrown and caught, nothing else. Now, by marking them with a suitable stereotype you can make these classes into first class citizens in your model; in other words, you make them appear as basic building blocks.

Stereotypes also allow you to introduce new graphical symbols for providing visual cues to the models that speak the vocabulary of your specific domain (see fig 4).

Graphically, a stereotype is rendered as a name enclosed by guillemots and placed above the name of another element (see fig 3). Alternatively, you can render the stereotyped element by using a new icon associated with that stereotype (see fig 4).

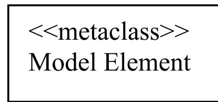

Fig 3. Named stereotype

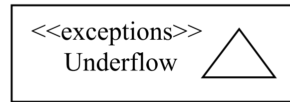

Fig 4. Named stereotype with icon

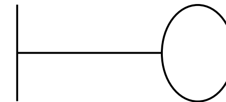

Fig 5. Stereotyped element as icon

Tagged Values

Tagged values are properties for specifying keyword-value pairs of model elements, where the keywords are attributes. They allow you to extend the properties of a UML building block so that you create new information in the specification of that element. Tagged values can be defined for existing model elements, or for individual stereotypes, so that everything with that stereotype has that tagged value. It is important to mention that a tagged value is not equal to a class attribute. Instead, you can regard a tagged value as being a metadata, since its value applies to the element itself and not to its instances.

One of the most common uses of a tagged value is to specify properties that are relevant to code generation or configuration management. So, for example, you can make use of a tagged value in order to specify the programming language to which you map a particular class, or you can use it to denote the author and the version of a component.

As another example of where tagged values can be useful, consider the release team of a project, which is responsible for assembling, testing, and deploying releases. In such a case it might be feasible to keep track of the version number and test results for each main subsystem, and so one way of adding this information to the models is to use tagged values.

Graphically, a tagged value is rendered as a string enclosed by brackets, which is placed below the name of another model element. The string consists of a name (the tag), a separator (the symbol =), and a value (of the tag) (see fig 6).

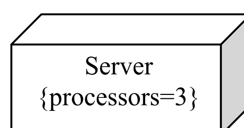

Fig 6. Tagged Value

Constraints

Constraints are properties for specifying semantics and/or conditions that must be held true at all times for the elements of a model. They allow you to extend the semantics of a UML building block by adding new rules, or modifying existing ones.

For example, when modelling hard real time systems it could be useful to adorn the models with some additional information, such as time budgets and deadlines. By making use of constraints these timing requirements can easily be captured.

Graphically, a constraint is rendered as a string enclosed by brackets, which is placed near the associated element(s), or connected to the element(s) by dependency relationships. This notation can also be used to adorn a model element's basic notation, in order to visualise parts of an element's specification that have no graphical cue.

For example, you can use constraint notation to provide some properties of associations, such as order and changeability (see fig 7).

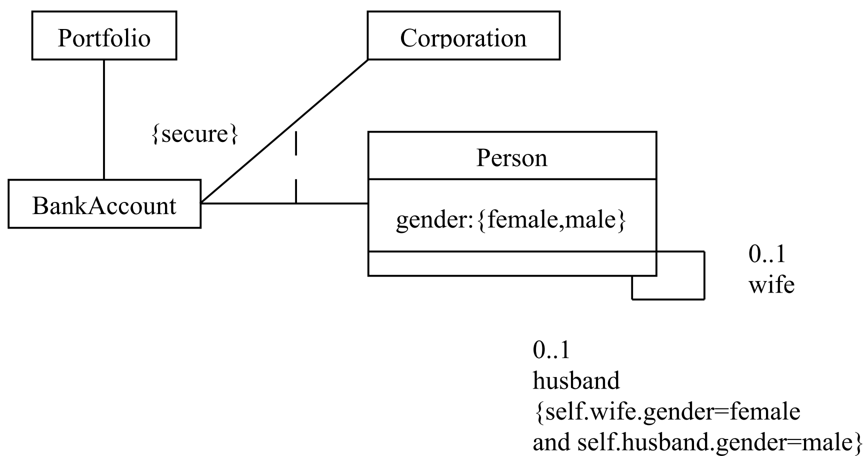

Fig 7. Formal constraint using OCL

Conclusion

Modelling is all about communication. When using the UML you are provided with all the tools you need in order to specify, visualise, construct, and document the elements of a software-intensive system. However, there will be circumstances where you might want to colour outside the lines, i.e. bend or extend the modelling language in order to shape and grow it to the specific needs of your project.

UML provides several extension mechanisms that allow you to do this without having to modify the underlying modelling language. These mechanisms let you add new building blocks, modify the properties of existing ones and even change their semantics.

The UML extension mechanisms provide not only a means for communication but also a framework for the knowledge and experiences of the individuals within a development culture such that the culture can evolve. They might not meet every need that arises within the development of a project, but they do accommodate a large portion of the tailoring and customising needed by most modellers in a simple manner that is easy to implement. However, it is imperative to keep in mind that an extension deviates substantially from the standard form of the UML and that by using it you might therefore encounter some interoperability problems. For this reason, it is essential to carefully weigh benefits and costs before using the extension mechanisms, and only do so when absolutely necessary.

References:

The Unified Modelling Language User Guide by Rumbaugh, Jacobsen and Booch [Addison-Wesley]
The Unified Modelling Language Reference Manual by Rumbaugh, Jacobsen and Booch [Addison-Wesley]
Using UML, Software Engineering with Objects and Components by Perdita Stevens and Rob Pooley