Modelling Software Development Using UML

Ananda Amatya

Department of Computer Science
University of Warwick
Coventry, CV4 7AL, U.K.
Email: Ananda.Amatya@dcs.warwick.ac.uk

Abstract. In Software Engineering, modelling is an activity that is applied to every software system throughout its life-cycle. UML is the OMG standard for building such models. UML models are used to support analysis, design and implementation of Object Oriented Software. UML provides features to extend it to model software for specialist application areas. The informality of diagrams and textual descriptions helps in its usability, while its fairly rigorous syntax and semantics minimise ambiguities in such models. UML extension mechanism using Profiles is discussed.
Key-words: Adornment, Annotation, Constraint, Profile, Stereotype, Tagged Value.

1 Introduction

This paper starts with an **overview** of how UML is used to build models in Software Development. **UML** [1] is the **OMG standard** [10] for describing the complete life cycle of **object-oriented** software in terms of models consisting of diagrams and texts. UML provides a whole range of diagrams appropriate for representing the different aspects of a software. It supports different **views** of the software **architecture**. In particular, UML provides features that are necessary to make visual models for the **Requirement, Analysis, Design, Implementation and Testing** phases of Object Oriented Software Development. Currently, UML is being used with different OOSD processes such as **Catalysis** [6] and **Unified Software Development Process** [9]. Tools for using UML for OOSD have been developed by **Rational** [16] and other vendors, and are increasingly being used.

2 Modelling

In Software Engineering, modelling is an activity that is applied to every software system throughout its life-cycle. Its purpose is to pinpoint relevant issues about the system at every level of **abstraction** needed from all points of view. In particular, models allow us to conceptualise by concentrating on the structure and behaviour of the system. Different views give rise to different models that capture different aspects of the system. UML is a **graphical** language with a standard syntax and semantics to formalise this modelling process. UML notation provides **rules and modelling elements** to document the whole software process in terms of diagrams and texts so that they may be used along with software **tools**.

3 UML

UML (**Unified Modelling Language**) provides a standard notation for visual modelling with well-defined syntax and semantics from **Logical**, **Component**, **Process**, **Deployment** and **Use case** points of view for systems development. Logical view point models deal with static structure and dynamic behaviour. Such models are used by Designers. Component view point models deal with organisation of codes. They are of interest to Programmers. Process view point models deal with concurrency, threads of control, communications and synchronisations. They are used by System integrators. Deployment view point models deal with the physical architecture of computers and devices and are used by System Engineers. Finally, Use case view point models deal with System functionality and are used by Analysts and Testers. UML can be used with all processes throughout the development life cycle and across different implementation technologies.

4 UML Notation

UML provides notation for its **Modelling Elements** (Basic elements, Composite elements, Relationships between elements, States and Interactions, Annotations), its **Modelling Mechanisms** (Adornments, Specifications), its **Modelling Rules**, its **Notation Extending Mechanisms** (Stereotypes, Tagged Values, Constraints). These combined with textual specification provide the diagrams and documents that form the models.

4.1 UML Modelling Element: Components

UML provides various **basic elements** for building models. Basic elements may be grouped into **composite elements**. **Relational elements** deal with various kinds of relationships between the model elements. Other elements are there to describe **object states** and **interactions**. **Annotations** are provided for clarifying the meaning. **Specifications** are saved in files to document responsibilities and capabilities of the model elements. They may also contain information on **Persistence** and **Concurrency**. Tools may be used to retrieve information from such specification files. **Adornments** make symbols mean specific things. In order to cover every possible situation UML provides notation extending mechanisms. One such mechanism is **stereotyping** which specialises the general notation to specific application areas. **Standard stereotypes** and **icons** are provided, though domain specific stereotypes and icons may be introduced as and when required. Notation extension uses **tagged values** to add more information. **Constraints** are used to show restrictions that apply. UML **Basic Modelling Elements** are Class, Object, Interface, Collaboration, Use Case, Active Class, Component, and Node. **Packages** are the only UML **Composite Model Elements**. They contain semantically related groupings of Classes, Components, Subsystems and Nodes. In **Classes Class names** are shown in boldface type, **Abstract classes** are shown in italic. In **Objects** (**class instances**) their names are shown by underlined class names. Class name or object name may be **elided** (anonymous). **Interfaces** that provide the connecting operations are indicated by lollipops. No **implementation** just **operations** are provided. **Collaborations** are indicated by dashed ovals. They **realize use cases**. Use case is shown as an ellipse; it shows a functionality of the system for the external user. **Active classes** are shown by thick bordered class boxes; they represent independent thread of processing. A **component** is a combination of one or more classes that forms a physical software. A **node** is a processor or device (a physical hardware). A composite model element is a **package** or a **subsystem** of base or composite elements.

4.2 UML Modelling Elements: Relationships

Relationships between UML Model Elements are represented by connecting lines, labels and arrows. **Association** (solid line between instances) or Aggregation (solid line with diamond (empty or full) between whole and part) occasionally directed, labelled, and with multiplicities and role names, **Dependency** (dashed line, possibly directed and occasionally with a label), **Generalisation** (solid line with hollow arrowhead pointing to the parent), and **Realization** (dashed line with hollow arrowhead, e.g., between interfaces and the classes that realize them and between use cases and collaborations that realize them) are the four different forms of UML Model Element Relationships. **Association** and **aggregation** refer to structural relationships between objects. In **normal** aggregation parts may be added or removed; parts may survive after the whole has died. In **containment** aggregation the parts are contained in the whole and die with the container. **Dependency** refers to relationships between whole classes; type of parameter passed to the calling class or the type of returned value. **Generalisation** is finding the superclass from subclasses, specialisation is the opposite. **Realization** is implementing classes that collaborate to realize the use case. Interfaces are realised by the classes or components that implement them. This goes through a **Refinement** process. For example, Analysis models are **refined** into Design models by adding design decisions like algorithms for operations.

4.3 UML Modelling Elements: States & Interactions

UML provides notation for Object **States** (Transitions, Events, Activities) and **Interactions** (Message exchange between objects, Action sequences, Links and

Solid line with a solid arrow head and a label giving the operation name). **Attributes** of an object change to give **transition** to another state. **Events trigger** the transition from one state to another. **Links** are instances of relationships and refer to links between objects. While Sequence and Collaboration diagrams focus on individual scenarios, State charts and Activity diagrams provide the behaviour of the particular objects throughout their life time.

4.4 UML Modelling Elements: Notes

UML provides notation for **Notes** (Rectangle with top-right corner turned and with textual **comment** inside, "**dog-eared**" box). Notes add information for **explanation** or **reminders**. Notes are associated to diagram elements using **dashed line**. Questions for future resolution are often shown as notes. Notes may contain **stereotypes**, e.g., `<<requirement>>`.

5 UML Modelling Elements: Diagrams

Majority of UML Diagram Modelling Elements are shown below in **Figure 1**.

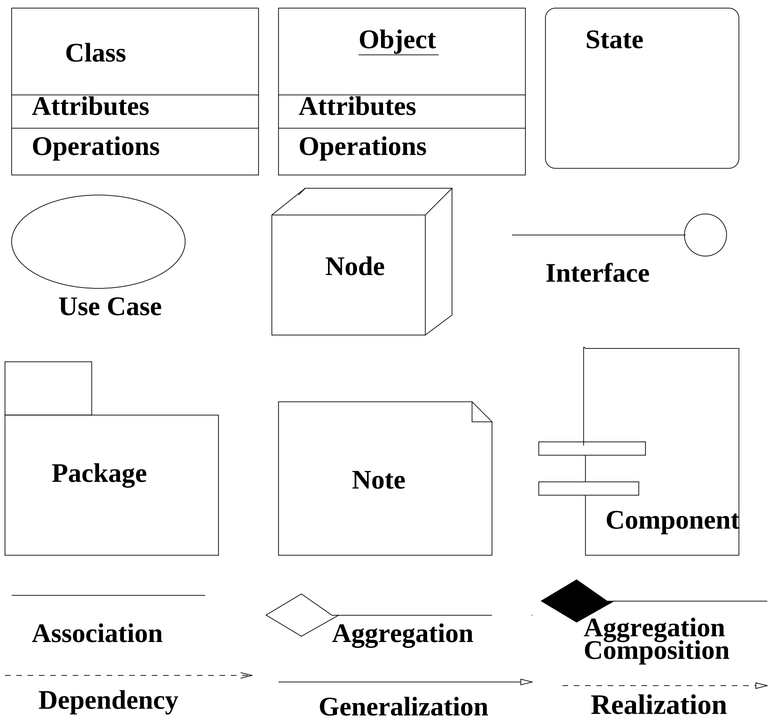

Fig. 1. UML Diagram Modelling Elements

6 UML Modelling Mechanisms: Adornments

Graphical adornments add **semantics**: **Boldface** (as in Class, Node type), **Underline** (as in object, instance of a node), **Multiplicities** of relationships. Graphs includes lines and texts. **Bold line rectangles** are used for **active classes**. **Italic names** are used for **abstract classes**.

6.1 UML Modelling Mechanisms: Specifications

Property of a model element may be specified further with a **name** and **tagged-values** (of predefined type, e.g. string or an integer). **Predefined properties** specify Documentation, Responsibility, Persistence, Concurrency. **Additional properties** (with new names and corresponding tagged-values) are given to specify further model element instances. These informal specification on model instances may be **accessed via tools**. For example, a Class may be given additional property of a Portfolio (name) with suitable specification (tagged-value) for it.

7 UML Modelling Notation Extension: Stereotypes

Stereotypes are used to **Extend/adjust** existing model elements. The **New** model element is based on an existing one. **Extra semantics** is added to the new model element. Typical model elements with **stereotypes** are: Class (e.g., <<Boundary>>, <<Entity>>, <<Control>>), Node, Component, Notes, Generalisation, Association (e.g., <<include>>, <<extend>>) and Dependencies. Standard (defined by the language) stereotypes may be used to modify an existing model element. Double angle brackets (<<...>>), "**guillemets**" are used to identify stereotypes. Graphical representation using special icons (e.g., **Actor**) may be used for stereotypes. Some standard stereotypes (all inside <<...>> : chevrons also called guillemets, pronounced Gee May):

```
<<become>>
<<constructor>>
<<copy>>
<<enumeration>>
<<exceptions>>
<<friend>>
<<helper>>
<<includes>>
<<process>>
<<send>>
<<query>>
<<type>>
```

7.1 UML Modelling Notation Extension: Tagged Values

Model elements with additional properties are given **name-value pairs**, the value being stored as tagged-values: string or integer to specify the property. Tagged properties are added on top of predefined ones. Some standard tagged values (all inside `{...}` :):

```
{version=...}
{abstract}
{author=...}
{status=...}
```

7.2 UML Modelling Notation Extension: Constraints

Constraints are use to **restrict** the usage or meaning or both of a model element. They may be declared in a tool and used in diagrams. Alternatively, they may be defined and applied as needed in a diagram. For example, association between Classes `SeniorCitizen` and `Person` is constrained to satisfy the condition that a person must be over 60 years of age in order to qualify as a senior citizen (**Figure 2**):

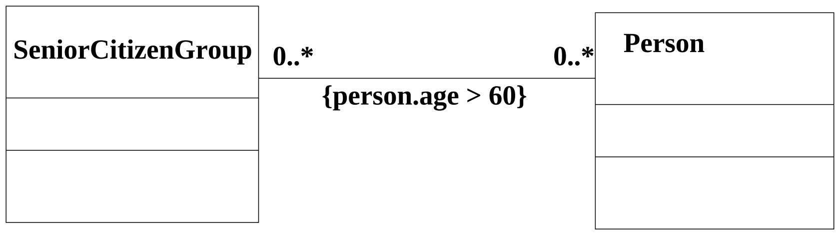

Fig. 2. UML Constraint Modelling Element

Some standard constraints (all inside `{...}`):

```
{disjoint}
{subset}
{overlapping}
{constraint1, constrraint2}
```

8 UML Modelling Notation Extension: OMG White Paper

The OMG White paper [14] suggests Lightweight & Heavyweight UML extensibility mechanisms. **Lightweight extensibility mechanisms** are the Stereotypes, Tagged Values and Constraints of the UML Standard described above;

they leave the meta-model for the UML Standard unchanged. **Heavyweight extensibility mechanisms** on the other hand extend UML meta model to allow new domain specific meta models. UML **Profiles** are to be provided to specialise a reference (standard) meta-model to a specific domain using extensibility mechanisms. Profiles provide the means of structuring, combining, filtering, selecting and projecting the extensibility mechanisms from one application domain to another. This will be of value in using UML in software development processes for diverse applications. Each domain specific UML meta model will have its own profile. A UML profile definition must specify the Profile's **Root Profile** (reference meta-model), **Name Space**, **Profile Generalisation**, **Profile Dependency**, **Profile Compatibility**, **Profile Exchange**,and **Profile Handling Mechanisms**. UML **Profile Standards** for Real Time, EDOC, CORBA, EJB, and many other application areas have been proposed and are at various stages of development. An early attempt to extend UML to Complex Real-Time Systems using preliminary ideas on profiles was reported by Brian Selic and Jim Rumbaugh [20]. The early Standard UML Profiles for **Software Development Process** and **Business Modelling** published with the UML Standard in June 1999 [10] used as their criteria **Stereotypes** (Association, Class, Collaboration, Model, Package, and Subsystem) and **Wellformedness Rules** (Generalisation, Association, and Subsystem Containment Constraints). The recently submitted Profiles for EDOC [8] (in response to the OMG RFP [12]) and CORBA [11] also use similar criteria. Desmond D'Souza et.al. [4, 5] use Catalysis [6] Packages (with package-import/generalisation mechanism) and Frameworks (to describe recurrent patterns in models and meta-models) as well as OCL [21] (to express constraints on the meta-model) to provide a mechanism for UML extension meta-model profiles. A related concept of **Prefaces** as extension mechanisms was proposed in [3]. Here again, *Prefaces* are structured using *Packages*. In response to OMG UML2.0 Request for Information [15] Tony Clark et.al., have proposed a kernel library to support Profile definition [2] [7].

References

1. Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modelling Language User Guide*. Addison-Wesley, 1998.
2. Tony Clark, Andy Evans, Robert France, Stuart Kent, Bernard Rumpe. *Response to UML 2.0 Request for Information, Submitted by the precise UML group*, http://www.cs.york.ac.uk/puml, December, 1999.
3. Steve Cook, Anneke Kleppe, Richard Mitchell, Bernhard Rumpe, Jos Warmer, Alan Cameron Wills, *Defining UML Family Members Using Prefaces*, in Mingins C and Mayer B (ed), Proc TOOLS 32, IEEE, 1999.
4. Desmond D'Souza, Aamod Sane, Alan Birchenough, *UML Profiles Considered Redundant*, 1999.
5. Desmond D'Souza, Aamod Sane, Alan Birchenough, *First-Class Extensibility for UML – Packaging of Profiles, Stereotypes, Patterns*, 1999.
6. Desmond F. D'Souza, Alan Cameron Wills, *Objects, Components and Frameworks with UML The Catalysis Approach*, Addison-Wesley, 1998.

7. Andy Evans, Stuart Kent, *Appendix A, A Kernel Library to Support Profile Definitions*, http://www.cs.york.ac.uk/puml, December, 1999.
8. Fujitsu Limited, Genesis Development Corporation, *UML Profile for Enterprise Distributed Object Computing – Joint Initial Submission*, Version 1.0, OMG Document ad/99-10-08, October 25, 1999.
9. Ivar Jacobson, Grady Booch, James Rumbaugh. *The Unified Software Development Process.* Addison-Wesley, 1999.
10. OMG, *OMG Unified Modelling Language Specification: Chapter 4. UML Standard Profiles*, Version 1.3, June, 1999. Available from http://www.omg.org
11. OMG, *OMG Corba Profile for UML*, OMG Document ad/2000-02-03, March, 2000.
12. OMG, *OMG UML Profile for Enterprise Distributed Object Computing RFP*, OMG Document ad/99-03-10, March, 1999.
13. OMG, *OMG UML Profile for Scheduling, Performance, and Time RFP*, OMG Document ad/99-03-13, April, 1999.
14. OMG Analysis and Design Platform Task Force, *Whitepaper on the Profile mechanism*, Version 1.0, OMG Document ad/99-04-07, April 1999.
15. OMG Analysis and Design Platform Task Force, *UML 2.0 Request For Information Version 1.0*, OMG Document ad/99-08-08, 25 Aug 1999.
16. Rational Rose 2000, Rational Software Corporation. Available from http://www.rational.com, 2000.
17. James Rumbaugh, Ivar Jacobson, Grady Booch, *The Unified Modelling Language Reference Manual*, Addison-Wesley, 1998.
18. Sinan Si Alhir, *Extending the Unified Modelling Language*, http://www.home.earthlink.net/ salhir, 1999.
19. Softeam, *UML Profiles and the J language: Totally control your application development using UML – White Paper*, 1999.
20. Brian Selic, Jim Rumbaugh, *Using UML for Modelling Complex Real-Time Systems*, March 11, 1998.
21. Jos Warmer, Anneke Kleppe. *The Object Constraint Language Precise Modelling with UML.* Addison-Wesley, 1998.