Department of Computer Science
UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS121 Lecture Notes: L02

Introduction to UML and DP

25 July 2014

Contents

L02.1 Introduction

This lecture discusses the Unified Modeling Language (UML). It is a visual modelling language used for software modelling and design in the object oriented paradigm of software development. After providing a brief history of the language, an overview of the components of UML 2 is provided.

Software design methods have strived to provide a way to model "good" design in software. The notion and background to Design Patterns is given, which provides a foundation for design.

L02.2 Unified Modelling Language (UML)

Prior to 1994, many people were busy developing modelling techniques and tools focussing on different aspects of object-orientation. In 1991, Rumbaugh *et al*, proposed the Object-modeling technique (OMT) which focussed on Object-oriented analysis (OOA). Grady Booch, while working at Rationale in the early 1990's developed the Booch method which focussed on both OOA and Object-oriented design (OOD). In 1992, Ivar Jacobson developed Object-oriented Software Enginering (OOSE). Rationale bought out the company for which Jacobson worked.

In 1995 Rumbaugh joined Rationale and started work on a modeling language along with Booch and Jacobson. The trio are fondly referred to as the *"Three Amigos"*. The mode;ing language they created unifies the respective languages they have independantly created earlier. They named this languge UML. UML 1.x was released in 1996 and included class diagrams, object modelling visualisations and use cases. In 2005, UML 1.4.2 was adopted as an ISO standard. 2005 also saw the Object Modelling Group (OMG)[1] adopting UML 2.x and taking responsibility for further developing UML [4].

L02.2.1 UML 2.x

From the beginnings in UML 1 of supporting 3 diagram types, UML 2 supports 14. Diagrams are divided into two broad categories, structural and behavioural. Figure 1 shows the diagram types supported by UML 2.

Structural diagrams depict what must be present in the system, being modelled. It shows a static view of the software being modelled. There are five types of structural diagrams namely class, object, package, component, and deployment diagrams.

Behavioural diagrams depict what the system being modelled must do. It models dynamic aspects of the software over time. There are four types of behavioural diagrams namely use case, state, activity and interaction diagrams. Interaction diagrams are further subclassed into communication, sequence, timing and interaction overview diagrams.

The diagram types highlighted in blue are relevant to this series of Lecture Notes and will be explained in more detail when required.

[1] `http://www.omg.org/gettingstarted/gettingstartedindex.htm`

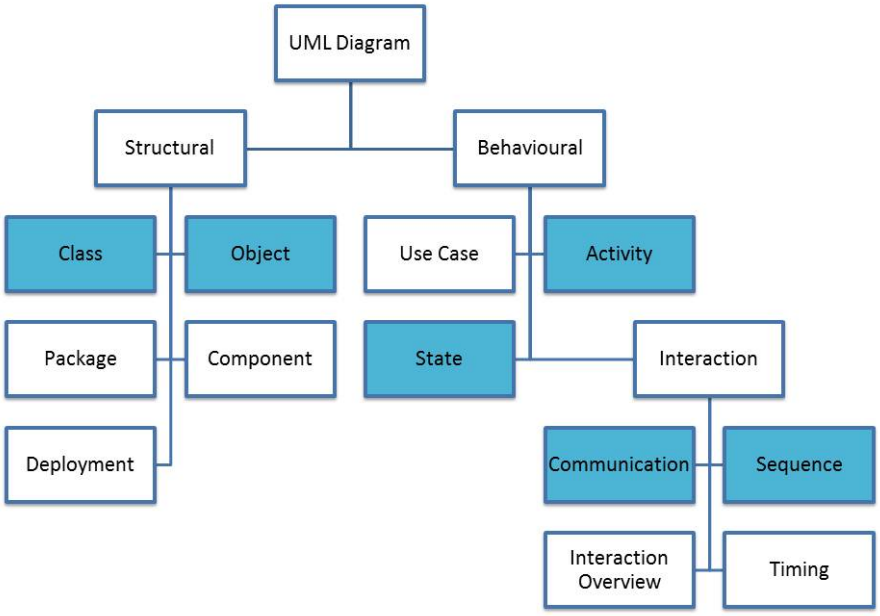

Figure 1: UML diagrams

L02.2.2 Class diagrams

A class diagram show the classes that an object-oriented system comprises of as well as the relationships between these classes. The internal structure of the class is described in terms of attributes and operations. The relationships are of two types, those that are related to the structure of the class and those that show the messages to be passed between the classes [2]. An example of a class diagram is provided in figure 2.

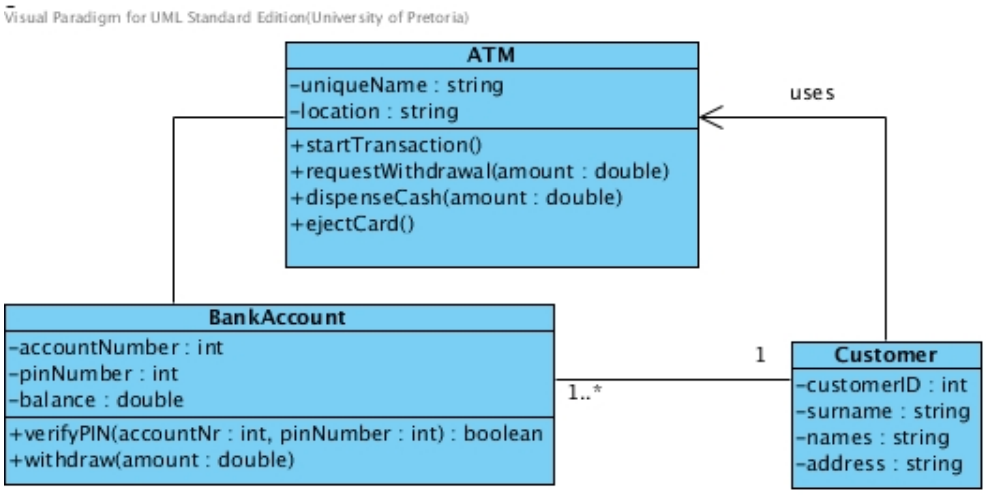

Figure 2: An example of a class diagram

L02.2.3 Object diagrams

Object diagrams are a special type of class diagram. An object diagram depicts the state of a system at a particular point in time. Object diagrams preserve the relationships between objects and show the current "state" or values of the attributes of the particular instances of the classes at a specific point in time. Figure 3 shows the state of the objects in the class diagram that was given in Figure 2 at a specific point in time whilea user called John Doe was doing a transaction on his savings account at a specific ATM in the Hatfield Plaza.

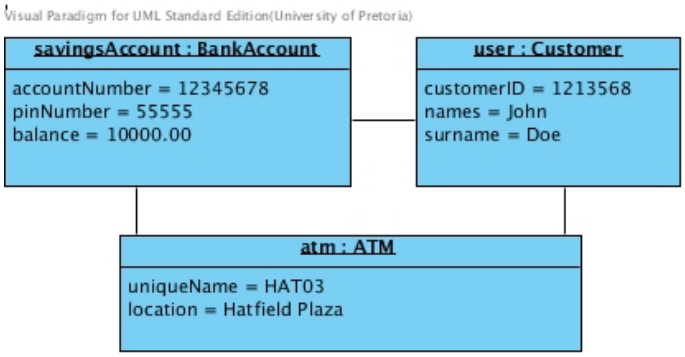

Figure 3: An example of an object diagram

L02.2.4 State diagrams

State diagrams illustrate process in terms of state changes. A state diagram models dynamic behavior of objects. It shows the changes in the state of an object. For example the states of an ATM can be 'Waiting', 'Connecting', 'Active', etc. Figure 4 shows that insertion of a card triggers the ATM to change from 'Waiting' to 'Connecting'. If successful it will change to the 'Active' state during which the user can perform various transactions, otherwise it will eject the card and return to the 'Waiting' state. When the user indicates that he/she wishes not to perform any more transactions, the ATM will eject the card and return to the 'Waiting' state.

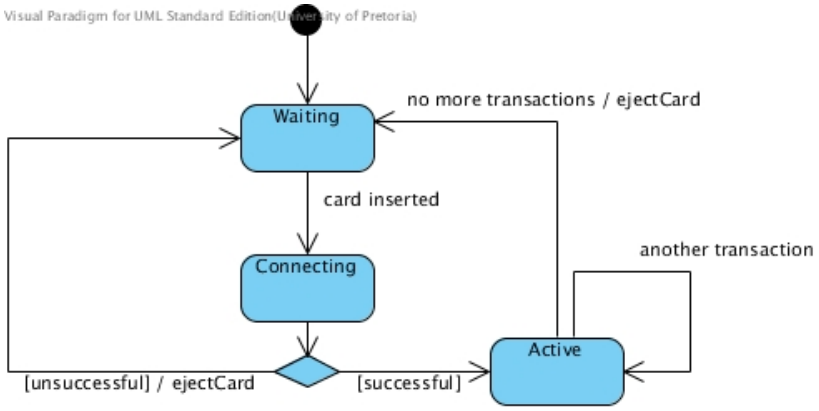

Figure 4: An example of a state diagram

L02.2.5 Activity diagrams

Activity diagrams illustrate process in terms of activities. An activity diagram is a kind of flow chart which shows the workflow behavior of an operation as a sequence of actions. For example the activities while an ATM withdrawal is processed can be 'insert card', 'enter PIN', 'check balance', 'eject money', etc. Figure 5 models the activities of a cash withdrawel process.

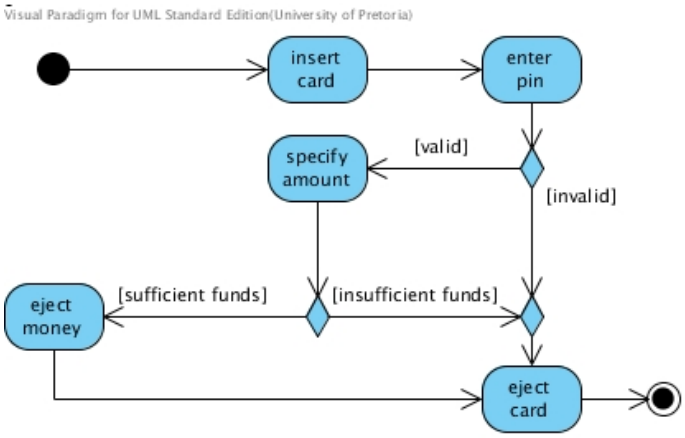

Figure 5: An example of an activity diagram

L02.2.6 Sequence diagrams

A sequence diagram is a type of interaction diagrams. Interaction diagrams model the interaction between objects, also referred to the messages passed between objects. Compared to the sequence diagrams in UML 1.x, the expressive power of sequence diagrams has been increased in UML 2.x. Sequence diagrams show interaction between objects. The order in which the messages are passed is visualised in the order of the communication lines in the diagram while all objects participating in the interaction are placed at the top of the diagram. Figure 6 shows some interactions between the objects shown in Figure 3 for a cash withdrawal transaction.

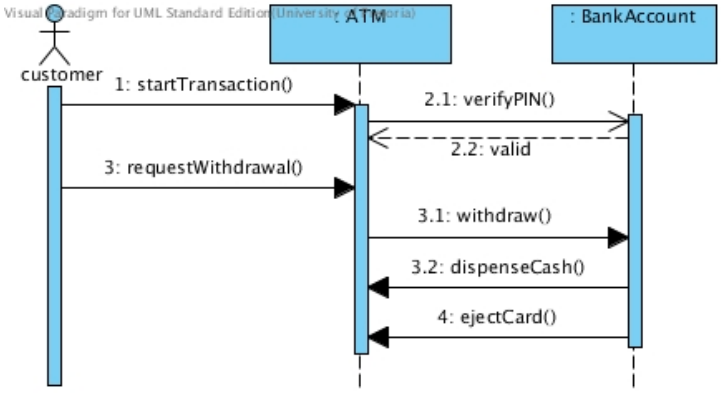

Figure 6: An example of a sequence diagram

L02.2.7 Communication diagrams

A communication diagram, like a sequence diagram, is a type of interaction diagram. Communication digrams was referred to as collaboration diagrams in UML 1.x.

The main distinction between sequence and communication diagrams is that sequence diagrams show interaction between objects over time while communication diagrams show the depth of the interaction between objects.

In communication diagrams the objects participating in the interaction are placed in proximity with one another to better visualise which objects are communicating directly with one another irrespective of the order in which the messages are passed. Figure 7 shows the same interaction that is shown in Figure 6.

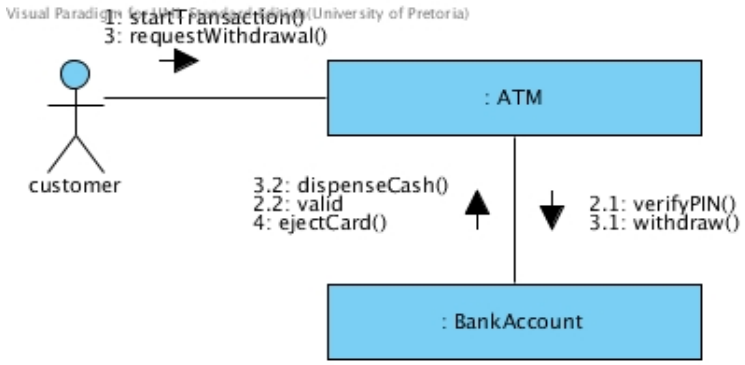

Figure 7: An example of a communication diagram

L02.2.8 Internet resources

The following internet resources may be helpful and provide a very good overview of UML.

- Wikipedia:URL: `en.wikipedia.org/wiki/Unified_Modeling_Language`
- OMG: URL:`www.uml.org`, refer to the document that describes the superstructure of UML
- Sparx System: URL:`www.sparxsystems.com/resources/tutorial`, presents a good overview of UML in the form of a tutorlal
- Tutorials Point: URL:`http://www.tutorialspoint.com/uml/uml_overview.htm`, another good tutorial on all the UML diagrams.

L02.3 Design Patterns

Patterns originated as an architectural concept when designing buildings. Christopher Alexander introduced this concept in his 1977 book on architecture, urban design, and community livability [1]. He proposed the following definition for a pattern.

> *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*

Design patterns in software design are no different. In software systems there are problems that occur over and over again for which design patterns can be identified to provide solutions to the problems. The idea of applying patterns to software was formalised by Kent Beck, of the Agile movement, and Ward Cunnigham and presented at OOPSLA in 1987.

The idea of software design patterns was originally not well received. It only started to gain popularity after 1994 when Gamma *et al* published their book of design patterns that describes simple and elegant solutions to specific problems in object-oriented software design [3].

The authors of the book is affectionately referred to as the "*Gang of Four*" or GoF. In the preface of the book, the GoF, states that the book neither introduces object-oriented programming and design, nor is it an advanced reference for object-oriented programming. They state that the book:

> *... describes simple and elegant solutions to specific problems in object-oriented software design. Design patterns capture solutions that have developed and evolved over time.*

The 23 classical design patterns, discussed by GoF, are categorised according to their purpose into Creational, Behavioural and Structural Patterns. A further level of categorisation applied has to do with the relationships between the classes. The object-oriented concept of delegation is classified as "object" while patterns with a predominantly inheritance relationship structure are referred to as "class" patterns.

When writing software it is important to address the quality attributes of the solution which includes user friendliness, effectiveness and efficiency. All patterns adresses one or more software quality requirements. Some patterns addresses specific quality requirements. For example Singleton and Flyweight addresses efficiency of memory usage while Prototype is aimed at reducing execution time.

Despite the fact that most patterns **increases** the execution speed owing to indirection created through excessive use of inherritance and delegation, invariably adaptability and maintainability of the code is greatly enhanced. Software applying design patterns is usually more robust and reliable owing to the use of tried and tested techniques. Design patterns are very useful abstraction tools that software engineers have at their disposal moving design decisions to a higher level of abstraction. Programmers who are competent in using design patterns are likely to be more effective in their work.

L02.3.1 Internet Resources

The following internet resources may be helpful and provide a very good overview of the classic design patterns.

- Wikipedia:
  - URL: `http://en.wikipedia.org/wiki/Software_design_pattern`, for an overview of design patterns
  - URL: `http://en.wikipedia.org/wiki/Design_Patterns`, for an overview of GoF [3]
- Huston: URL: `www.vincehuston.org/dp/`
- OODesign: URL: `www.oodesign.com`

References

[1] Alexander C, Ishikawa S, and Silverstein M (1977) *A Pattern Language: Towns, Buildings, Construction (Cess Center for Environmental)*. Oxford University Press, later printing edn.
URL `\url{http://downlode.org/etext/patterns/}`

[2] Bennett S, Skelton J, and Lunn K (2001) *Schaum's Outline of UML*. UK: McGraw-Hill Professional.

[3] Gamma E, Helm R, Johnson R, and Vlissides J (1994) *Design patterns : elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley.

[4] Wikipedia (2012). Unified Modeling Language — Wikipedia, The Free Encyclopedia. [Online; accessed 28-July-2012].
URL `\url{http://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=504223112}`