UML: Abstraction as a Criterion for Defining Class Diagrams and Object Diagrams

Ivan Pogarcic[1], Miro Francic[1] and Vlatka Davidovic[1]

[1] Business Dept, Study of Information Systems, Polytechnic of Rijeka
Trpimirova 2/V, 51000 Rijeka, Croatia
{pogarcic, mfrancic, vdavid}@veleri.hr

Abstract. UML is undisputedly the most efficient and effective tool of information systems analysis and design. Abstraction as paradigm, represent the basis of an object-oriented approach to development of information system and software solutions. No matter what background team members have (i.e. information technology or problem domain experts), the capability of abstraction is of crucial importance, especially at the early phase of the system's structure identification. Since class and object diagrams specify system's structure, indicating how to identify and relate them, they are an issue for system analysts and designers. This paper examines the following issue: to what extent abstraction level influences the need for creating object diagrams in shaping class diagrams and vice versa to what extent concretization at the level of an object diagrams influences structural decomposition of class diagrams.

Keywords: abstraction, concretization, class diagram, object diagram, UML

1 Introduction

Development of information system is a complex process consisting of different activities structured in phases which can be undertaken linearly or iterative. Phases follow life cycle which roughly can be identified as: system requirements elicitation (analysis phase), designing the system (design phase) and implementation phase [4].

Number of factors influences the accepted approach to development (linear or iterative) but one of the most relevant for the specific project is how the results of the previous phase can be used in the next step. Object modelling and design well support this issue than traditional structure analysis and design.

Because of complexity of today's information systems due to complex distributed hardware, different software, more complex networks and comprehensive data bases, design and development of such systems are very demanding.

Furthermore, information system developing process comprises different activities which presume engagement of experts with different knowledge related to problem domain (business) and information technology. Participation of business users of information system is of crucial importance, especially in early stages of development. One of the main issues in developing process is communication among team members. Modelling is a tool for overcoming this issue. Models are main

deliverables in analysis and design phase. Understanding reality in a more simplified way is aim of modelling. Abstraction, the process of neglecting some aspects of reality not interesting in the context and stress those relevant to the context, is a wide accepted method in developing conceptual and logical models.

Modelling is a complex process. Real system and model requires precise undertaken specification of system elements, their adjustment visualization, adequate construction and detailed documentation. All these activities are executed with purpose of ensuring complete and clear overview of system's structure and its functioning mode.

If we adopt the following goals of modelling: simplified visualization of real world (problem domain), adjustable specification of system structure and its behaviour, possibility of making discrete records necessary for individual phases of system development, adequate documentation of activities and decisions during development process, ways to improve design quality, reducing errors and avoid ambiguity, improving communications, separate different concerns in managing complex systems development, allowing hierarchical modelling, facilitates impact analysis of requirements and design changes and supports incremental development[3], UML is today's best tool for achieving these goals.

For the purpose of this document, only class diagrams and consequently object diagrams out of thirteen UML diagrams will be taken into account, especially from the point of view how abstraction influences the class and object diagrams definition.

2 Relational approach vs. OO (object oriented) approach

Technology improvement supported by theoretical results of information science researches and better user's education in information technology, imply changes in putting information system elements together. Rapid changes in business result in growing users requirements regarding their number and complexity. The way of handling data and information has significantly changed, too. Development of accurate information system tailored to fulfil user's requirement, demand adequate methodology, tools and technology. For data storing and manipulating them, three main possibilities are available today: Relational Data Base Management System (RDBMS), Object-Relational Data Base Management System (ORDBMS) and Object-Oriented Data Base Management System (OODBMS).

Additionally, but not sufficiently changed the approach to design and development of information systems. Methodologies in fields of analysis and design changed. Basic advantage of OO programming in comparison to procedural is a modularity implying written program modules do not have to change when new object types are being added. Features of existing programs are retained and if necessary new are being added. By leaving procedural programming and turning to object-oriented programming programmers primarily wanted to make these activities more effective and efficient.

Consequently, changes should have been implemented both in methodology, technology and tools that would adequately support this approach.

Since UML is declared as language it has syntax and semantic rules. Set of symbols comprise its notation. However, UML is implementation free and its aim is to support object oriented analysis and design.

UML notation encompasses important elements of object-oriented systems (classes, objects, methods etc). UML possesses a special terminology for individual model parts excluding ambiguity. If we consider UML model as tool its important parts are: elements, notation, terminology, views, packages and naturally adequate model documentation that usually presents obstacle for processes of design and system development.

Formally UML as tool is comprised out of 13 diagram forms (depending upon UML version) divided in two groups: behavioural diagrams and structural diagrams. Another language SysML System Modelling Language (SysML) is a new extension of UML modelling, but it has the same goals as UML and from the OO point of view SysML follow UML (implementation of inheritance paradigm) [2]. SysML represents a subset of UML 2.0 with extensions needed to satisfy the requirements of the UML for Systems Engineering RFP [2]. It comprises: Activity Diagram and structural diagrams group where Block Definition Diagram and Internal Block Diagram are being introduced that in lower abstraction level have Parametric Diagram subordinated while on higher level Requirement Diagram was added.

Software development with UML is characterized by: use-case orientation, focussing on basic element's features, on their behaviour and dynamics of system, on domain problem concepts and iterative and incremental ability.

System analysis deliverables, among others are set of use cases. Traditional structure methods use functional decomposition instead of use cases. Exceptions required additional activities in system analysis and design. In addition, UML support unique view through system development process and offer better change management due to iterative and incremental approach. The same model done in some development phase can be refined in the next phase without the change in the basic shape.

3 Why abstraction is important?

What has actually changed with OO approach in design and programming so far? Primarily, four paradigms were promoted as basis of this approach.

The first paradigm Abstraction is a mechanism for representing reality by neglecting some details focussing on some other aspects of interest. The essential, inherent aspects of an entity are a result of abstraction. In system design, this allows designers focussing on object state and its behaviour [1]. Abstraction allows to analysts in developing models to define entities without details**.** Analogy with mathematical abstraction is more evident in programming while from the philosophical point of view it is more evident in analysis and design. Although other OO paradigms: Encapsulation, Inheritance and Polymorphism are important, abstraction can be observed as starting and the most important paradigm.

Modelling by UML allows creating model in different abstraction level, from the low to the high level. But, abstraction as paradigm is not inherent characteristic of all

team members. On the contrary, some team members, primarily business users, can consider it as disadvantage. Because of the mandatory participation of end users in system development their ability of abstraction is of a great importance. Business users usually describe their business through exceptions, so designers should have a high level of abstraction capability to generalize users' requirements. In fact, the issue is how to compromise business users' and designers' views. As a result could be the functionally very rich system, covering majority of users' specific requirements, or pure system if the designers did not identified properly users needs.

It seems that user ability of abstraction is more important then of other team members because modelling of the system structure is based on the abstraction of properties of system elements, while system element behaviour demonstrate concretization of those properties.

Building up the model using UML through different levels of abstraction is necessary also in implementation phase using some of the OO programming language (C++, Java, Pearl etc).

4 Abstraction and Class diagrams

Class is a category or group of entities that have similar features and mode of operation. Class is a kind of classifier whose features are attributes and operations. Classes reflect system entities and there are relationships among them. When two classes are in relationship, each of them has specific role in that relationship.

Class Diagrams describe target system by the objects and classes and their relationships. They reflect static system structure. It provides a wide variety of usages; from modelling the domain-specific data structure to detailed design of the target system. Abstraction in class and class diagram definition provides different possibilities for designers. As an example are abstract classes enabling more design possibilities but, their usage depends on abstraction capability of a designer.

Being class diagrams a result of the system analysis based on use cases, the abstraction is indispensable in that phase, too. Regarding this, each exception could become particular use case resulting in huge number of use cases. So, abstraction can decrease number of use cases, while concretization increases it.

Class diagram has different semantics in system analysis and system design. In system analysis phase higher abstraction is allowed and recommended. In system design phase classes are more detailed described and new attributes and new methods are added so abstraction level becomes lower. Sense of measure in applying abstraction becomes common problem for system user and designer.

Abstraction is important both for class definition, their association (responsibility) and inheritance. Abstract class can not instant object but is important as for generating classes that will be inherited. Inheritance and abstraction are of great importance in modelling. However, inheritance causes specific kind of relationship - hierarchy. This hierarchy kind has obligatory character just in purpose of overtaking class features of inherited class while implementation of features gives freedom in adjustment.

In syntax of Java program language, inheritance is emphasized with reserved word "extend" opposite to interface implementation marked as "implements". Interface

improves abstraction and we can imagine it as a complete abstraction class. Interface ensures form but not realization and it tells us how implementation classes will look like.

5 Abstraction and Object diagram

Object is an instance of class in a particular point of time[5]. Object diagrams are instances of class diagrams. Accordingly, class diagrams can initiate numerous object diagrams While the class diagrams are design phase deliverables, we do not deal with classes during the system usage, instead we use objects. The question is, why do we need object diagrams if they completely instance class diagrams?

Object diagrams are useful for exploring “real world” examples of entities and the relationships between them. Although class diagrams describe reality quite well, many people find them too abstract - object diagram are better for explaining complex relationships between classes. We find object diagrams as a tool for describing real world at a very low abstraction level. In the case the description of the class become too complicated due to a high number of attributes and operations, object diagrams are very useful to understand the reality by "learn by example".

If we look at class diagram it is not obviously how classes cooperate between themselves because objects are those who carry activities and collaboration in a particular point of time. An object diagram shows the relationship between the instantiated classes and the defined classes, and the relation between these objects, from the logical view of the system. These are very useful to explain smaller portions of the system, when the system class diagram is very complex, and also sometimes recursive.

Object diagrams are convenient in analyzing and defining specific problems with specific classes and it can give answers for specific situations: What if class becomes more complicated? What if the number of attributes is too high? What if methods become immense? Object diagram is adjustable for explanation of exceptions caused by objects with specific features.

Notation in object diagram is simpler and includes only objects and relationship among them. When specific object is being displayed it is necessary to ensure enough information for its full transparency. UML object diagrams are effectively notational subsets of UML communication diagrams, although object diagrams are used to explore structure whereas communication diagrams explore behaviour. It is common for object diagrams to evolve into communication diagrams simply by adding messages to the diagram.

6 Conclusions

Reality-based modelling and visualization (a picture is worth a thousands words) is extremely helpful in performing analysis and design. UML is probably the one and only wide-spread modelling tool / language that meet those purposes. Like other tools, it also has disadvantages and encounters problems in practice. We are likely to

expect that these identified problems should soon be solved. SysML is an example of this.

Another problem that should be emphasized is the adoption of defined paradigms and specific tools and languages by designers, and especially by end users that will depend exclusively on their own aptitude and capability. Abstraction is a cognitive process that demands intellectual capabilities of an individual. The problem is that the quality of the project's output depends upon team members' personal skills. For a successful work on the project with UML, the recent literature suggests, among other things, the following [7]: focussing on artefacts; implementation of various abstraction levels in building Analysis model, Design Model and Implementation Model ; choosing of appropriate detail level; identifying of individual modelling style; focussing on re-usable system elements.

With this paper we tried to underline the abstraction as a paradigm and the necessity of giving adequate attention to it that, consequently, refers also to other OOP/OOM paradigms since they rely upon it. Creating use case and class diagrams is the most critical part of UML modelling. The application of object diagram is inevitable in describing exceptions. Researches presented in [7] show that class diagrams vs. object diagrams usage ratio is 9:3.6, what means more then 2.

References

[1] Blaha, M., Rumbaugh, J Object–Oriented Modeling and Design, with UML (2nd Edition), Prentice Hall, (2004)

[2] Bock. C., "SysML and UML 2.0 Support for Activity Modeling", Journal of International Council of Systems Engineering, vol. 9 no.2, pp. 160-186 2006.

[3] Booch, G., Rumbaugh, J., Jacobson, I.:Unified Modelling Language User Guide, Addison-Wesley, 1998

[4] Leszek, A. Maciaszek: Requirement Analysis and System design – Developing Information Systems with UML, Addison-Wesley, 2001

[5] Miles, R., Hamilton,K.: Learning UML 2.0 , O'Reilly , Media (June 1, 2006)

[6] OMG Unified Modelling Language Specification - UML 2.1.1. Superstructure Specification

[7] www.magicdraw.com/training