A Systematic Approach for Constructing Static Class Diagrams from Software Requirements

Nabil Arman
Department of Mathematics and Computer Science
Palestine Polytechnic University
Hebron, Palestine
Email: narman@ppu.edu

Khalid Daghameen
Department of Electrical and Computer Engineering
Palestine Polytechnic University
Hebron, Palestine
Email: dkhalid@ppu.edu

Abstract: The trend towards the use of object-oriented methods for software systems development has made it necessary for the use of object-oriented approaches in object-oriented software systems development. Static class diagrams represent an essential component in an object-oriented system design. The development of such class diagrams in a systematic way is very crucial in an object-oriented development methodology. The paper describes a new approach for obtaining these static class diagrams in a systematic way, which is very essential in object-oriented development practice.

Keywords: Object-Oriented Analysis and Design, and Class Diagrams.

1. Introduction

Many object-oriented development methodologies have been developed in the past two decades. These methodologies include Object Modeling Technique (OMT), OOSE, OPEN [1, 2, 3]. Commercial methodologies that can be purchased are available today, including Rational Rose and Rational Unified process. [4, 5]. A system development methodology (SDM) has been defined as "*…a systematic approach to conducting at least one complete phase (e.g., requirements analysis, design) of system development, consisting of a set of guidelines, activities, techniques and tools, based on a particular philosophy of system development and the target system*" [6]. From this definition, it is generally recognized that there is a great emphasis on "being systematic" for any phase of system development methodology.

The requirements engineering practice using object-oriented development methodologies has been well-documented in the literature [7, 8, 9]. In [7] a number of case studies were presented. In these case studies, the use of a systematic approach for obtaining static class diagrams was absent. In [8], the role of object-oriented process modeling in requirements engineering phase of information systems development is presented. In [9], object-oriented requirements engineering and management is explained.

These approaches are categorized into two main categories according to Edward Yourdon. The two categories are evolutionary to Object-Oriented analysis and Design and revolutionary to Object-Oriented analysis and Design. Rambuagh and

Martin-Odell are two approaches of evolutionary approach. Booch and Wrifs-Brock are two approaches of revolutionary approach [10,11,12].

Some approaches to Object-Oriented analysis and design are based on use cases after the introduction of UML. Dough Rosenberg and Matt Stephen proposed an approach for Object modeling based on use cases [13]. Their approach includes the static and dynamic models of Object-Oriented, while our approach in this paper is concerned with static models. In addition, their approach depends mainly on software engineers/practitioners, whereas our new approach is less dependent on human intervention.

This paper presents a systematic approach consisting of a set of guidelines and techniques for obtaining static class diagrams from software requirements. The guidelines are explained in detail and a motivation case study is given to illustrate the approach. Research into a systematic approach for obtaining class diagrams practices is therefore needed to improve understanding of object-oriented system development methodologies in general and class diagrams in particular.

The remaining of the paper is organized as follows: section 2 presents the main principles and guidelines of our approach. In section 3, a case study is presented in which the new approach is used to construct the static class diagram. Finally, section 4 presents the conclusion.

2. Static Class Diagrams Construction

A static class diagram is constructed from software requirements and problem statements by following the steps below:

1) Identify the list of all nouns and noun phrases from the problem statement. These nouns and noun phrases represent classes and adjectives represent attributes. A software engineer/practitioner obtains such a list from the problem statement, use cases, actor-goal list or application narrative description [10,11,12,13].

2) Make a refinement for the nouns determined in step 1. Such nouns represent values for attributes, or nouns that are identical to other nouns. This step is important to ease the process of having the static diagram, but it is not necessary; nouns that are not related to the problem statement are isolated classes in the static class diagram. This means that they are not part of the system. Nouns that are identical according to the problem statement appear in the static class diagram with the same attributes and associations as other nouns. From that one also needs to pick one of such classes. For example, in a problem statement concerning a university, a teacher may be called by a teacher, or a professor or an instructor. Such nouns mean the same thing in this problem statement [10,13].

3) Arrange the nouns from step 2 into a matrix keeping the order of such nouns the same in the rows and columns. The nouns that appear in the left column is called “left cell” in the paper, and the ones that appear in the top row is called ”top row”. In this early step, one keeps the nouns and noun phrases at the top of the list and the adjectives at the bottom of the list.

4) Fill up the table with the letters (I, H, P, U or keep it empty); by answering a YES/NO questions. The four questions are: Is a “left cell “ a “top row”?, Has a “left cell” a “ top row” ?, Is a “left cell” part-of a “top row”? and Does a “left cell” use a “top row”?, consecutively. It is important to notice that once a cell is occupied by a letter, you don’t need to do the rest of the questions for that particular cell.

5) The answers of the four questions must be according to the problem statement, use cases, actor-goals or application narrative description. In a university problem, the answer for a question such as “Is the graduate-student a student?” is yes. The answer for “Has Student information a date?” is yes. On the other hand, an answer for the question such as “Is a student a graduate-student?” is no, since there is a student who is not a graduate student for that problem domain.

6) Identify those nouns which are classes and those which are attributes. Any row that has an “H” letter is a class, as for those rows which don’t have an “H” letter is left to the software practitioner to identify. Rows that have more than one “I” is a class, also the row which has a single “I” is also a class. As for the adjectives, all of them are attributes [10,12,13].

7) Rearrange the nouns again in the table taking into account that all candidate classes must be sorted according to the number of “H” letters in each row and the others are kept to the end of the list for the classes, while the attributes and methods should occupy the tail of the list. Fill up the lower half of the matrix with letters by asking the simple four questions again.

8) Each identified class in the table should have a class diagram. If you are using UML models, a stand alone class is drawn by the name of the class [14,15,16].

9) Identify the hierarchy by connecting classes using the cells containing “I” letters. An L-shape is formed between two candidate classes. For the hierarchy stair-shape like is obtained by forming more than one L-shape from those “I” cells.

10) An UML static class diagram is obtained from the information you obtained from the previous step.

11) For the “P” cells, arrange them into groups according the groups of “I” cells. For example, a stand alone “P” in a row means that this is an attribute “left cell” is an attribute in the class “top row”. For two “P” cells in a row, find the equivalent of “I” cells that form the lower part of L shape from a row in the top. Those are part of the lower L-shape, and this attribute is part of the class which is in the higher L-shape of “I” cells. A group of three “P” cells is the same and you need to find the Upper L-shaped class upon identifying the Attribute position of an attribute.

12) For those “P” cells, which are classes, or those classes that have “has-a” associations with another class. The first step is to locate the position of the

class which should be connected to another class. This is done using the same as step 11.

3. A Case Study

Assume that the following text/problem statement is taken from a software requirements document:

Vessels are one kind of containers that contain liquids, vessels are different in shapes and sizes. Each vessel has a capacity which is the maximum quantity of the substance that is contained using a vessel. The amount of substance in a vessels wobbles between zero to the maximum of the capacity of a vessel. When someone fills up a vessel, he/she is going to add some substance to the existing substance in the vessel.

A lot of shapes of vessels are available, but we are concerned with few of them, some of them which are rectangular tanks, this type has a rectangular base and a height, the rectangular base is identified by its length and width. A cubic tank which is the same as a rectangular one except that the base is a square base and both length and width have the same value. A cylindrical tank has a circular base and height. A circular base is identified by its radius.

To construct a static class diagram form the above problem statement, the steps are undertaken as follows:

- The first step in analyzing the problem statement is to identify a list of nouns and noun phrases, such as vessel, container, size, capacity, tank, rectangle, square, height, length and width.

- The next step is to take some of those nouns out of the list. Such nouns are those which are identical or they refer to the same thing. In the problem statement, tank, vessel and container are synonyms of the same thing. So vessel is picked up. Some nouns are not part of the problem although they are mentioned in the problem statement. A software engineer/practitioner must take them out of the list such as substance and liquid. In this problem, liquid is not part of the problem. Even if the software engineer/practitioner chooses not to take the liquid out. Such a class would be an isolated class in the static class diagram.

- All names that are generated from the previous step are arranged in an adjacent matrix as in figure 1.

- Fill up the table (below the diagonal) with letters (I, H, P), to have the letter I. The answer for the following question must be yes, Is “Left cell” a “top row”?. For example “Is a rectangular tank a vessel? “. The answer is yes. Therefore, the cell that corresponds to rectangular tank with vessel should have an “I”.

- If the answer for a question is no, the cell remains empty. Like “Is a capacity a vessel?”. The answer is no, therefore, the cell remains empty.

- Move to the next question which “is part-of” question. A question is generated to have an answer either yes or no. The question is “Has (left row) a (top cell) ?” or “Is the (left row) part of a (top cell)?”. If the answer to any of them is yes, then the corresponding cell is marked with a “P”.

- The next question to fill up the table is “Has (left row) a (top row)?”. If the answer to this is yes, write an “H” in the cell corresponding to the “left row” and “top cell”. In the example, some cells are changed to “H”.

- The table now becomes like figure 1. Any row that has a P letter means that this is an attribute. A row which has a single “I” is a class. Others would be a class if the row is an adjective. That means it is an attribute. If it is a verb, then it is considered a method [10,13].

	Vessel	Capacity	Amount	Rectangular tank	Cubic tank	Cylindrical tank	Length	Height	Width	Radius	
Vessel	I										C
Capacity	P	I									A
Amount	P		I								A
Rectangular tank	I	H	H	I							C
Cubic tank	I	H	H	I	I						C
Cylindrical tank	I	H	H			I					C
Length				P	P		I				A
Height				P	P	P		I			A
Width				P					I		A
radius						P				I	A

Figure 1. Object-Oriented Relationship Information

- With a pencil, form the English letter L by connecting I-cells together. You need to form a stair-shape so you can have the hierarchy form of the class diagram. The first path is (Vessel → Rectangular Tank, Rectangular Tank → Cubic Tank) and the next path is (Vessel → cylindrical Tank) as shown in figure 2.

- UML shapes of the hierarchy are drawn as shown in Figure 3 [14,15].

- Rearrange the table so that the classes are at the top and the attributes are at the bottom and apply the same for all of them as shown in Figure 2. The rearrangement should take into consideration the number of “I” cells within each row. The rows should be sorted in ascending order.

- For the "P” cells, form a region of one cell, two cells, three cells, four cells and so on.

- The two attributes capacity and amount can be in the top of the class hierarchy as you form a rectangular region of four cells and that region intersects with the top hierarchy which is vessel. This region covers the lower part of all L-shape.”, such attribute must be at the top of the hierarchy.

- Length has two boxes and should be in the Rectangular tank class as in figure 2.

- Height has two regions, one with two boxes and the other with one as we can’t form a region with three boxes that forms a lower part of L-shape that are related in the hierarchy. From this, one can notice that one is in the rectangular tank class and the other in the cylindrical tank class.

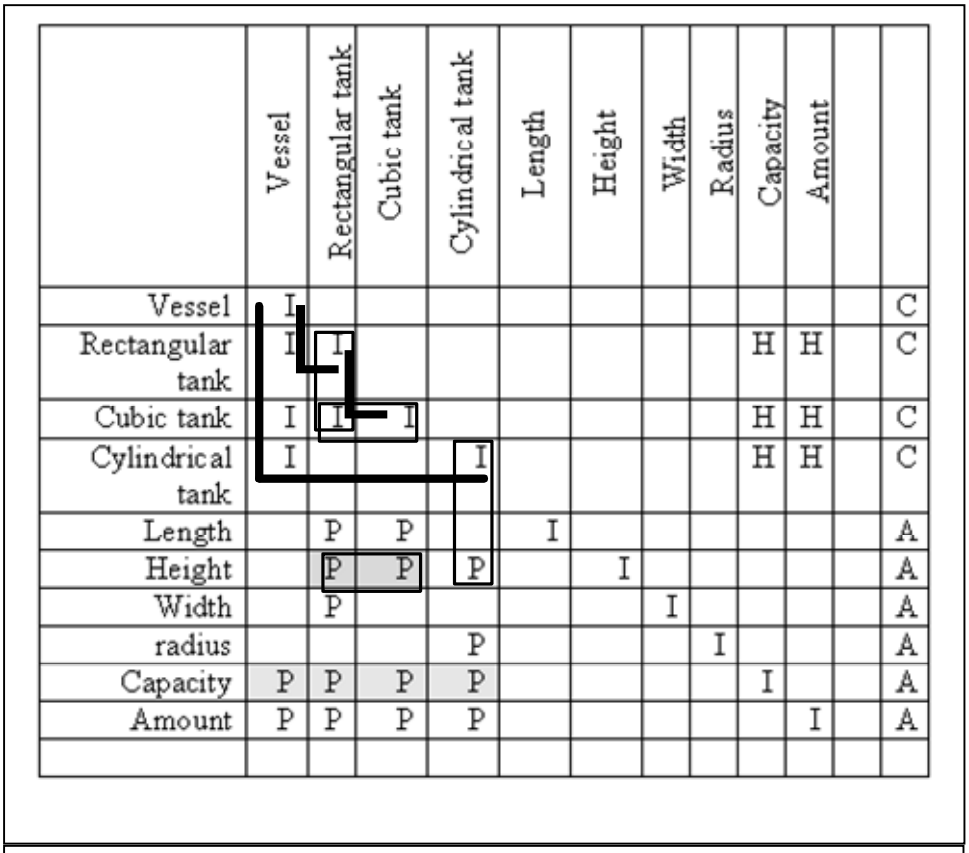

Figure 2. Class Relationship

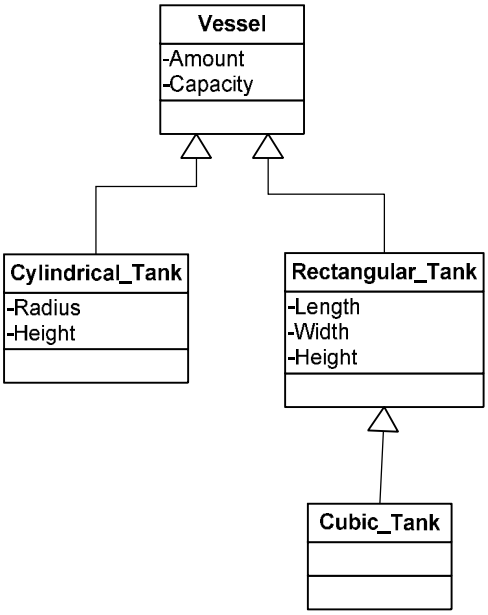

Figure 3. UML Static Class Diagram

4. Conclusions

In this paper, a systematic approach to construct static class diagrams from software requirements is presented. The principles and guidelines of the approach are explained in detail and a case study is given to demonstrate the applications of these principles to a real-life example.

References

1. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., "Object-Oriented Modeling and Design," Prentice-Hall, Egnlewood Cliffs, NJ, 1991.

2. Jacobson, I, Christerson, M., Jonsson, P., and Overgaard, G., "Object-Oriented Software Engineering: A Use Case Driven Approach," Addison-Wesley/ACM, NY, 1992.

3. Henderson-Sellers, B., and Simons, A., "The Open Software Engineering Process Architecture: From Activities to Techniques," Journal of Research and Practice in Information Technology, Vol. 32, No. 1, pp. 47-68, 2000.

4. Quatrani, T., "Visual Modeling with Rational Rose and UML," Addison-Wesley, Reading, MA, 1998.

5. Jacobson, I., Booch, G., and Rambaugh, J., "The Unified Software Development Process," Addison-Wesley Longman, Reading, MA, 1999.

6. Wynekoop, J., and Russo, N., "Studying System Development Methodologies: An Examination of Research Methods," Information Systems Journal, Vol. 7, pp. 47-65, 1997.

7. Dawson, L. and Darke, P., "The Adoption and Adaptation of Object-Oriented Methodologies in Requirements Engineering Practice," ECIS 2002, June 6-8, Gdansk, Poland, 2002.

8. Kasser, J., "Object-Oriented Requirements Engineering and Management," Systems Engineering Test and Evaluation (SETE) Conference, Canberra, October 2003.

9. Knott, R., Merunka, V., and Polak, J., "The Role of Object-Oriented Process Modeling in Requirements Engineering Phase of Information Systems Development," EFITA 2003 Conference, 2003.

10. Roper Pressman, “Software Engineering: a practitioners approach”, Mcgraw hill, 5th edition, 2000.

11. Shari Pfleeger, Joanne Atlee, “Software Engineering: Theory and Practice” , 3rd edition, Pearson Prentice Hill, 2006.

12. Simon Bennett, Steve McRobb and Ray Framer, “Object-Oriented Systems Analysis and Design Using UML”, 3rd edition, McGraw Hill, 2006.

13. Doug Rosenberg and Matt Stephens, “Use Case Driven in Object Modelling with UML: Theory and Practice”, Apress, 2007.

14. Desmond Francis D’Souza, Alan Cameron Wills, “Objects, Components, and Frameworks with UML: The CatalysisSM Approach”, Addison Wesley Longman, 1999.

15. Kim Hamilton, Russull Miles, “Learning UML 2.0”, O’really, 2006.