Interaction Driven OOAD

D. Janaki Ram*

Department of Computer Sc. & Engg.,

Indian Institute of Technology Madras,

Chennai 600036, Tamil Nadu, INDIA.

Phone: +91-44-22574354

Fax: +91-44-22574352

djram@lotus.iitm.ac.in

Arun Kumar†

IBM India Research Lab.,

Block-1, Indian Institute of Technology,

Hauz Khas, New Delhi 110016, INDIA.

Phone: +91-11-26861100

Fax: +91-11-26861555

kkarun@in.ibm.com

Abstract

Traditionally, object-oriented analysis and design (OOAD) approaches have focussed on identification, analysis and specification of individual objects in a given system. Design Patterns brought a shift in this paradigm by introducing the approach of identifying design solutions that involve a group of classes/objects rather than individuals. However, the potential of this paradigm has not been fully realized yet. The aim of recent research has mainly been restricted to identification of new design patterns from successfully implemented solutions of the past. In this paper, we propose to utilize the concept of interactions among classes and objects as a key factor during problem analysis phase and as a natural way of designing object oriented systems. We look at different techniques that could be used to enable such an analysis and design process and identify open issues in those directions.

Keywords: Design Patterns, OO anaylsis, interactions, automated composition, object responsibility.

*Contact author

†This work was done when the author was at Indian Institute of Technology Madras.

1 Introduction

Design Patterns [1] brought a paradigm shift in the way object oriented systems are designed. Instead of relying on the knowledge of problem domain alone, design patterns allow past experience to be utilized while solving new problems. Traditional object oriented design (OOD) approaches such as Booch [2], OMT [3], etc. advocated identification and specification of individual objects and classes. Design Patterns on the other hand promote identification and specification of collaborations of objects and classes.

However, much of the focus of recent research has been towards identification and cataloging of new design patterns. The effort has been to assimilate knowledge gained from designing systems of the past, in various problem domains. The problem analysis phase has gained little benefit from this paradigm. Most projects still use traditional object oriented analysis (OOA) approaches to identify classes from the problem description. Responsibilities to those classes are assigned based upon the obvious description of entities given in the problem definition. Pattern Oriented Technique (POT) [4] is a methodology for identifying interactions among classes and mapping them to one or more design patterns. However, this methodology also uses traditional OOA for assigning class responsibilities. As a result, its interaction oriented design phase (driven by design patterns) receives its input in terms of class definitions that might not lead to best possible design.

The missing piece here is the lack of an analysis method that can help in identifying class definitions and the collaborations between them which would be amenable to application of interaction oriented design. There are two key issues here. First is to come up with good class definitions and the second is to identify good class collaborations.

It has been observed in [5] that even arriving at good class definitions from the given problem definition is non-trivial. The key to various successful designs is the presence of abstract classes (such as an event handler) which are not modeled as entities in the physical world and hence do not appear in the problem description. In [5], anticipating change has been proposed as the method for identifying such abstract classes in a problem domain. Another difficult task is related to assignment of responsibilities to entities identified from the problem description. Different responsibility assignments could lead to completely different designs [6]. Current approaches such as Coad and Yourdon [7], POT etc. follow the simple approach of using entity descriptions in

the problem statement to define classes and fix responsibilities. We propose to follow a flexible approach towards assigning responsibilities to classes so that the best responsibility assignment can be chosen.

The second issue is to identify class collaborations. Techniques such as POT analyze interactions among different sets of classes as specified in the problem description. Such interacting classes are then grouped together to identify design patterns that may be applicable. However, as mentioned earlier, only the interactions among obvious classes are determined currently. Other interactions involving abstract classes not present in the problem or interactions that become feasible due to different responsibility assignments are not considered. We present some techniques that enable the designer to capture such interactions as well.

The rest of this paper is organized as follows. Section 2 gives an example to motivate the problem being handled. Section 3 describes various techniques to enable interaction driven analysis and design. We discuss open issues in section 4. Finally, we conclude in section 5.

2 A Motivating Example

In this section we present an example adapted from [6] to motivate the need for an interaction based analysis and design approach. Consider a simple system of borrowing books from a library. The library operates under the following rules:

1. Only registered personnel can borrow books.
2. Based upon her category, each user has a limit on the number books she can borrow.
3. Multiple copies of same book cannot be borrowed.
4. At least one copy of the book requested should be available with the library that is not loaned to any user and has no reservations on it.

We consider two possible designs of such a library system. Both the designs have four classes, namely, *LibrarySystem*, *User*, *BookDetails* and *BookCopy*. In the first design, LibrarySystem class has the core responsibility and acts as a coordinator to execute the book loaning workflow. It first authenticates the user (rule 1) and then verifies from BookDetails whether the book requested is

available for loan from the library (rule 4). It then checks with the User whether she is authorized to borrow that book (rules 3 and 4, i.e. checks whether maximum books allowed limit is crossed and that it is not already borrowed). Finally, it obtains a reference to a BookCopy object, invokes *borrow* method on that and updates the User object.

In the second design when a *borrow* request is issued, the LibrarySystem object authenticates the user (rule 1). Once done, it invokes *borrow* operation on BookDetails object corresponding to the requested book. BookDetails object in turn verifies rule 3 and 4 and obtains reference of one particular BookCopy object. It then invokes *borrow* on the User and passes the reference to BookCopy object. User object verifies rule 2 and invokes *borrow* on the BookCopy object.

The first design maps the physical world and such designs can generally be obtained, using traditional OOA techniques, from the problem description. The second design, however, assigns responsibilities to classes in a way that is not reflected in the problem description. As a result, we have a different design that distributes the responsibilities across various objects rather than centralize in one large monolithic object.

From an interaction point of view, the classes in the second design follow Proxy [1] design pattern. The actual access to the BookCopy object is through different proxy objects each of which does some housekeeping task. The LibrarySystem object performs authentication, the BookDetails object verifies whether a copy of the book is available and so on.

From a maintenance point of view, the second design appears to be better than the first one. This is because any change/addition in rules concerning users, books etc. would be localized to individual classes. Whereas in the first design the main LibrarySystem class would always get modified requiring re-verification of the remaining portion of the heavy class. From reusability point of view, the second design results in more functionally cohesive classes that perform a specific function. These are more amenable to reuse than the all-purpose LibrarySystem class of first design.

From the above example, it can be observed that class responsibilities and interactions between classes play a major role in the design of object oriented systems. In the next section, we present techniques that can increase the influence of these two important factors in the analysis and design phases of OO systems.

3 Interaction Based Analysis and Design

In this section we present two approaches for enabling interaction driven design for object-oriented systems.

3.1 Top-down approach

This approach is applicable to situations where the designer knows the solution to the given problem. It is true for problem domains that have well established high-level solutions and different implementations vary in low level details (for e.g. Enterprise Resource Planning (ERP) systems). Her main concern is to realize that solution in a way such that the implemented system has nice properties such as maintainability and reusability etc.

To achieve this goal, the system designer selects appropriate design patterns that form the building blocks of her solution. Having obtained this design template (*design type*), she maps the classes and objects participating in those patterns to the entities of the problem domain. This mapping implicitly defines the responsibilities of various classes/objects that represent those entities. To help clarify the concept, consider a scenario where an architect is assigned the task of building a flyover. Flyover construction is an established science and the architect knows the solution to the problem. She starts by identifying component patterns such as road strip, support pillars, side railings and so on. Having done that, she maps the participating objects to actual entities in the problem domain. This would involve defining the length and width of the road strip based upon the space constraints specified in the problem. The height and weight of the pillars get decided based upon the load requirements specified. The entry and exit points get decided based upon the geography of the location and so on.

This results in a concrete *design instance*. Some new classes or objects, not existing in the domain model, may also have to be introduced for a successful instantiation of the design template. For instance, the problem domain may not model an abstract entity such as an *event handler* which may be a participant in some portion of the design template. Such generic classes/objects may be drawn from a common repository of *utility classes*.

Interaction driven analysis phase here is simple since the interactions (in the form of design patterns) are already well established and directly obtained from the knowledge base.

3.2 Bottom-up approach

This approach is applicable in scenarios where interactions in the problem domain are not well understood and need to be discovered and explored.

This situation is a fundamental problem faced by the designers of object oriented systems. It relates to the fact that object oriented analysis (OOA) does not help much in creating a solution to the problem at hand. The analysis phase is mainly concerned with enhancing the understanding of the problem domain. This knowledge is then later used by a problem solving approach to come up with a solution possessing good design properties. As a result, at the end of the analysis phase the designer has a set of well defined components that need to be assembled together for realizing a solution. For instance, to build a *route finder* application the OOA phase helps in modeling the domain objects such as roads, vehicles, cities, addresses etc. but does not actually provide a solution for finding routes between two given addresses. This is similar to having various pieces of a jigsaw puzzle but the puzzle still needs to be solved.

The problem in software systems is further complicated by the fact that there is generally no unique solution to a problem. There are always trade-offs at various stages and the resulting designs are a reflection of the choices made at those stages. In the jigsaw puzzle example this is similar to the situation where different sets of the same puzzle are available each differing from the another in terms of the design of its component pieces. Some component designs may help in solving the puzzle faster and more efficiently than others.

The bottom-up approach helps in such situations where the entities in the problem domain have been identified by traditional OOA techniques but multiple choices exist in terms of assigning responsibilities to those entities. Unlike top-down approach, the mapping of responsibilities to entities is not dictated by the design solution specified by the designer. Instead, the task of the designer here is to try various responsibility assignments and create an *interaction specification* involving those objects. The objective of this interaction driven analysis is to obtain an interaction specification that helps in arriving at a solution with best design characteristics possible.

Having identified the entities in the domain, the starting point for the designer is to identify various alternatives available for assigning responsibilities to individual objects. Her domain knowledge helps her in this task. Given these alternatives for potential object definitions and standard

utility objects (such as schedulers, event handlers etc.), the next step is to find compositions of these building blocks (i.e. interactions of these objects) that provide alternative solutions to the problem. This task is a non-trivial one especially when done manually. There are just too many combinations to be considered, for any human designer to obtain alternative solutions in a reasonable amount of time.

We need to apply (semi-)automated software composition techniques based on some formal specification. Several such approaches have been recently investigated in the context of e-services. These include workflow based approaches and AI Planning based techniques [8, 9]. Other formal techniques for specifying composition include Petri-net based models [10], automata-based models, temporal logics etc. from verification community and XQuery, XML constraint tools based techniques from data management community [11].

The resulting candidate compositions (i.e. interaction specifications) then need to be compared with existing design patterns either manually or automatically. It is not beyond imagination to visualize that with advancement in automated composition techniques, new design patterns may get identified during this process. For instance, techniques such as Reinforcement Learning have resulted in new novel solutions in various domains such as playing Backgammon [12]. In such a case, the resulting designs may need to be evaluated manually.

The best design among the alternatives is then chosen for implementing the system.

4 Open Issues

4.1 Identifying interactions

This is a crucial step in the analysis phase and the success of remaining phases depend on it. The issue here is to identify interactions which are not evident from the problem description but may hold the key to an efficient design solution. The bottom-up approach proposed in this paper takes a step in this direction but a lot more work is needed. The analysis method should be such that it is able to incorporate abstract classes such as event handlers, proxies etc. Moreover, current analysis methods map entities to responsibilities of individual classes in terms of services they provide and methods they invoke on other classes. However, an entity may be realized by a set of classes. For instance, an adapter class hides the interface of an adaptee class and they collectively

provide the desired functionality. Similarly, an abstraction and its implementation provide a single functionality through separate classes resulting in increased maintainability. The analysis method needs to be able to determine when is it appropriate to realize an entity responsibility by means of multiple interacting classes.

4.2 Representation of Class Responsibilities

Since we need to specify different alternative class responsibilities, as in bottom-up approach, a mechanism is required to document them in a machine interpretable format. Some of these responsibilities would get captured in the form of methods a class exports or methods it invokes on other classes. However, other responsibilities with respect to its interaction with other classes needs to be explicitly specified. These may include pre- and post-conditions for different method invocations, other properties such as '*hasSameInterfaceAs* <another class>', '*hidesInterfaceOf* <another class>' etc. Languages such as [13] could be used as it is or extended for this purpose.

4.3 Language for Specifying Design Patterns

The approaches for OO Design proposed in this paper favour automatic techniques over manual ones for reasons described earlier. This means that we need a mechanism to be able to express design patterns in a format amenable to be read and interpreted by programs. Some attempts have been made at defining such pattern description languages [14, 13]. One of these or some variation of these could be used to express design patterns in a formal language.

4.4 Comparison of Software Designs

Once we have alternative designs available, they need to be compared to arrive at the best one. Each design may consist of multiple design patterns. The criteria here would not be to simply count the number of design patterns used but to evaluate the interaction between patterns and also between other design elements used. This would involve an understanding of good and bad design interactions and an ability to identify them in a given design. The final challenge would be to do it automatically.

5 Conclusion

In this paper, we proposed to use interactions among collaborating classes as the basic unit of analysis and design of object oriented system. We believe that this methodology of analyzing possible interactions to come up with a solution to the given problem and then fixing appropriate class responsibilities would lead to good OO designs. This is because the design phase, in this methodology, is not restricted by pre-specified responsibilities and also takes into account abstract classes and non-straightforward interactions between classes. We presented two approaches applicable in different problem scenarios and identified some open issues for further research.

Acknowledgements

We thank Vinay Kumar Reddy for his reviews and comments on the paper.

References

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. *Reading: Addison-Wesley*, 1995.

[2] Grady Booch. Object-Oriented Analysis and Design with Applications. Addison-Wesley Professional, 1993.

[3] James R Rumbaugh, Michael R. Blaha, William Lorensen, Frederick Eddy, and William Premerlani. Object-Oriented Modeling and Design. Prentice Hall, 1990.

[4] D. Janaki Ram, K. N. Anantha Raman, and K. N. Guruprasad. A pattern oriented technique for software design. *SIGSOFT Software Engineering Notes*, 22(4):70–73, 1997.

[5] Wayne Haythorn. What is object-oriented design ? *Journal of Object-Oriented Programming*, 7(1), March-April 1994. pp. 67-78.

[6] Leonor Barroca, Jon Hall, and Patrick Hall. Software Architectures. Springer-Verlag London Limited, 2000, pages 87-99.

[7] Peter Coad and Edward Yourdon. Object Oriented Analysis. Prentice Hall PTR, 1990.

[8] Jinghai Rao and Xiaomeng Su. A Survey of Automated Web Service Composition Methods. In *Proceedings of First International Workshop on Semantic Web Services and Web Process Composition*, July 2004.

[9] Vikas Agarwal, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A Service Creation Environment based on End to End Composition of Web Services. In *Proceedings of Fourteenth International World Wide Web Conference (WWW), Chiba, Japan*, May 2005.

[10] Rachid Hamadi and Boualem Benatallah. A Petri net-based model for web service composition. In *Proceedings of the Fourteenth Australasian database conference on Database technologies*, pages 191–200, 2003.

[11] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. E-Services: A Look Behind the Curtain. In *Proceedings of the International Symposium on Principles of Database Systems (PODS), San Diego CA, USA*, June 2003.

[12] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning:An Introduction. MIT Press, Cambridge, MA, 1998.

[13] Object Management Group. UML 2.0 OCL Specification. http://www.omg.org/docs/ptc/03-10-14.pdf.

[14] Rajeev R. Raje and Sivakumar Chinnasamy. eLePUS - a language for specification of software design patterns. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 600–604, 2001.