An Application Domain Based on General Object Oriented Software Models

Buthainah Fahran Al-Dulaimi
College of Education for Woman, University of Baghdad, Iraq,
buthynna@hotmail.com

Abstract:

Any software application can be divided into four distinct interconnected domains namely, problem domain, usage domain, development domain and system domain. A methodology for assistive technology software development is presented here that seeks to provide a framework for requirements elicitation studies together with their subsequent mapping implementing use-case driven object-oriented analysis for component based software architectures. Early feedback on user interface components effectiveness is adopted through process usability evaluation. A model is suggested that consists of the three environments; problem, conceptual, and representational environments or worlds. This model aims to emphasize on the relationship between the objects and classes in the representational model and the elements in the considered system. Implementing this model on some practical examples is investigated and resulted into promising improvement in software design and understanding.

Key words: Object Oriented Modeling, Software Modeling, Software Domains and Software Architecture.

مجال تطبيقٍ مستندٍ إلى نماذج برامج كائنية المنحى عامة

بثينة فهران الدليمي
كلية التربية للبنات/ جامعة بغداد

المستخلص:

يمكن تقسيم أي تطبيق برامج في اربعة مجالات مختلفة مترابطة وهي نطاق المشكلة، ونطاق الاستخدام ونطاق التنمية ونطاق النظام. ويراد من منهجية التكنولوجيا المساعدة لتطوير البرمجيات هنا أن تسعى إلى توفير إطار لدراسة الاستنباط مع متطلبات التعيين في وقت التنفيذ. واعتمدت الملاحظات في وقت مبكر على واجهة المستخدم من خلال تقييم فعالية مكونات عملية الاستخدام. ويقترح الانموذج الذي يتكون من البيئات الثلاث؛ المشكلة، المفاهيمية، وبيئات التمثيل أو العالمين. هذا النموذج يهدف إلى التأكيد على العلاقة بين الكائنات والطبقات في النموذج التمثيلي والعناصر في النظام. وقد نفذ هذا النموذج على بعض الأمثلة العملية تم التحقيق فيها وأدت إلى تحسين واعد في مجال تصميم البرمجيات والتفاهم.

1. Introduction

Object-Oriented software methodologies provide notations and guidance to model both the static structure of the program and the dynamic behavior of the objects. These methodologies assist programmers to analyze, understand, design, visualize, and document artifacts in software systems. Any problem environment is the observed domain or the real world even if it is virtual one. It consists of phenomena that can be classified by conceptual methodology through obstruction resulting into the construction of a model environment, called conceptual environment. Phenomena are observed in the problem environment and classified into concepts in the same environment[1]. This environment includes the modeler and the perspective applied to the real world. Domain elements in both problem environment and model environment are tightly connected. The perspective alone determines to what extent the model should reflect certain aspects of the problem environment, i.e. to what extent it represents the real world. In the model environment, descriptions are applied to model concepts and representations are applied to model phenomena. It might be noted that any description serves as a specification and a representation serves as an instance[2].

Software modeling can be defined as a mapping technique for a set of phenomena and concepts on a real environment to corresponding objects and classes in a model environment while the reverse process is called interpretation. Tippell and Oregan[2] used object oriented model for associative technology software development. User interfaces and object oriented model were implemented by Thomson[3]. Support for intuition, exhibiting and collaboration in case tools for creative OO modeling was tackled by Damma et al.[4]. Component introduction as a conceptual architecture model is suggested by Nowack[5]. Concept and language mechanism in software modeling was considered by Jacobsen[6]. Perry and Wolf[7] introduced theoretical and practical aspects of software architecture which are then broadly adopted in industry[8]. The architecture model expresses the developer's conception of the system architecture in order to understand the system model from some perspectives. S/w development processes turned into architecture centric either for dealing with complexity, risk management or effective resolution of quality attributes. Hofmeister et al.[9] reported a generalized model of software architecture design that maintain implicitly or explicitly, a backlog of smaller needs, issues, problems they need to tackle and ideas they might want to use. This backlog includes architecture assets, ideas, context, constraints and architecturally significant requirements besides evaluation results. It drives the workflow from architectural analysis to architectural synthesis and architectural evaluation helping the architect to determine what to do next.

Hofmeister et al.[9] and Falessi et al.[10] reported that software architectures can be built following Software Architecture Design Methods (SADM), which mainly consist of three major activities: requirement analysis, decision making and architectural evaluation activities, as depicted in Fig 1.

Fig 1. General s/w Architecture Design Method[10]

However, no SADM is precise enough to encode all details on how software architecture must be manipulated when performing an activity. Furthermore, Attribute-Driven Design ADD method[11] follows a recursive design process where it starts by choosing an element of the system for decomposition, then identifying candidate architectural drivers followed by design

concepts that satisfy these drivers and finally instantiate architectural elements and allocate responsibilities. Model-Driven Engineering and Aspect-Oriented techniques is reported by Perovich et al.[12] to achieve systematic and assisted construction of the software architecture of Enterprise Applications.

2. Modeling Development Process

Madsen[1] identified and modeled four different closely related domains in software development process; problem domain, usage domain, development domain and system domains. Problem domain, as the subject of information, is explicitly modeled as user understanding into usage domain. This usage domain monitors and controls the information of the problem domain leading to the development domain. The development domain contains the software developer, vision, development platform, reusable software parts, developer's experience and all the activities performed by the developer. Finally, system domain is the design and implementation artifacts that describe and constitute the system. It contains everything needed to design the system architecture sub-models and eventually run the system.

The developer's conception of the system architecture is expressed thoroughly in the architecture sub-models in order to represent detailed descriptions to be used in the implementation of any Domain Specific programming Language DSL later on. Any software model or language must aim at meeting some essential success factors specific to the use of that language. Without consideration of general success factors, such as commitment from higher management or the availability of skilled staff, the essential success factors are; (1) Learnability[13]: Developers have to learn an extra language and stay up-to-date. (2) Usability[14]: Easy and convenient tools and methods supporting the language, (3) Expressiveness[15]: Using a DSL, domain specific features can be implemented compactly, (4) Reusability[16]: Possible reuse at model level of partial or even entire solutions, (5) Development costs[17]: The DSL helps developers to model domain concepts where corresponding source codes are generated automatically, reducing development costs and time, (6) Reliability[13]: automation of large parts of the development process leads to fewer errors.

After a brief introduction and definition of software modeling and software architecture design in sections 1&2, section 3 outlines the proposed software modeling system. Implementation and results are included in section 4. Then section 5 concludes the paper.

3. Proposed Software Modeling Systems

Modeling is the activity of connecting concepts in the referent system with the concepts in the model system. From the above mentioned studies, a general software model can be thought of by breaking any project understudy into three worlds or environments; problem, conceptual and representational environment. Problem environment comprises the situation row problem status. Conceptual environment comprises the conceptual models of the system and the conceptual framework on which the conceptual model is founded. Representational environment contains a representation of the conceptual models and the representational tools used to build representational models. Hence, the modeling process involves the raw problem environment, the developing efforts to constitute the conceptual environment and the problem execution. Modeling can be achieved in the following stages.

a. Mapping real environment: Fig 2 illustrates the mapping of real environment tackled by the developer who observes the phenomena of the problem environment, applies the conceptual action needed producing descriptions that allow to setup specifications. Therefore, one may define programming in this context as the process of abstraction in problem envi-

ronment, whereas the abstraction in model environment is considered of three sub-functions; the identification of phenomena and their properties, classification, and composition. Elements of the problem environment such as objects and their properties represent phenomena.

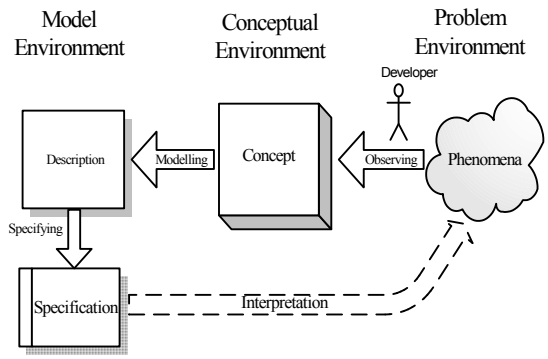

Fig 2. Mapping Real System

The main concern of any modeling efforts is the programming and execution of specific obstruction process of this environment satisfying a systematic use conception to describe and deal with the phenomena. Modeling in this context means putting the precise descriptions for the observed phenomena which then results into specifications. Finally interpretation of the resulting specifications for the produced model environment must is expected to represent the phenomena.

b. Modeling real environment: Particularly when modeling intangible phenomena we need to distinguish between phenomena in the system and phenomena in conceptual model. The central points are the abstraction processes and the notions of concept and phenomenon. When modeling, a set of entities in the problem environment may be considered as part of the system, forcing the system developer to make delimitation of what constitutes the system and of what constitutes the environment of the system. The developer defines the system, makes framework and conceptual models of the system, and represents the models. A modeling process will be an iterative process in which the modeler iterates over the elements of the modeling process, as illustrated in Fig 3.

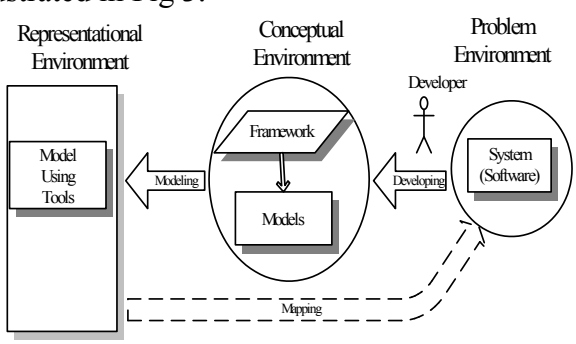

Fig 3. Modeling Real system

The system is defined by selecting entities in the problem environment and to regard them as parts of the system. The conceptual models are formed on the basis of some chosen conceptual framework then, representations of the conceptual models are expressed using suitable notation. This definition demarcates which part of the problem environment is to be modeled.

The framework of the conceptual environment assumes the kind of entities in the proposed system specifies what kind of abstractions can be made over the system when forming the conceptual models and determines how the system is conceived. While the used notation for the representational environment determines how the conceptual model is represented and therefore it influences how the system is conceived too. Furthermore, the modeling process may have added complication by a role called problem owner. This situation can be considered problematic as the problem owner motivates a certain system demarcation and a certain modeling perspective, which specifies the system-aspect that is important to portray in a model. The conceptual framework and the representational tools are chosen on the basis of their suitability for the system-aspect to be modeled[6, 18, 19].

c. Software Modeling: During modeling software process, the problem environment constitute software domain, which means that a specific considered system consists of software in a broad sense, i.e. program code, diagrams, or running programs are represented by some notation. Hence, investigating software from the perspective of the notation in which it is expressed, it can be understood in terms of its immediate constituents. Therefore, an immediate conception of the software is obtained due to the fact that software is based on certain notation. However, immediate conception of software by developers is not enough for its reasoning; therefore abstractions are needed in order to cope with the inherited software complexity.

d. General Object Oriented Software Model: Object-oriented modeling makes the assumption that systems are conceived in terms of phenomena and concepts. When making a conceptual model of the system, the model developer identifies phenomena in the system and creates corresponding phenomena in the conceptual world. A phenomenon in the conceptual environment is a manifestation of the fact that a part of the system is regarded in a specific way. The properties that the model developer associates with the phenomena in the conceptual world are dependent on the modeling perspective, which in turn is determined by the problem owner.

The elements of the problem environment are modeled into interrelated concepts and phenomena in the conceptual model by the developer abstraction process. The properties associated with the phenomena in the conceptual environment used to reason about the system. These phenomena and the concepts are then represented by objects and classes in the representational environment called the object oriented model. The object model helps us to reason about the system being modeled by exposing a specific aspect of the system as properties of the objects in the object model. Fig 4 illustrates the relationship between the object model and the system.

Any objects in the object model of the representational environment needs to be associated with specific phenomena in the problem environment, therefore object models are considered, examined in system then update actions in object model. Each object model contains two different parts; a reference to the phenomenon it models and the phenomenon information, which consists of object properties and relations.

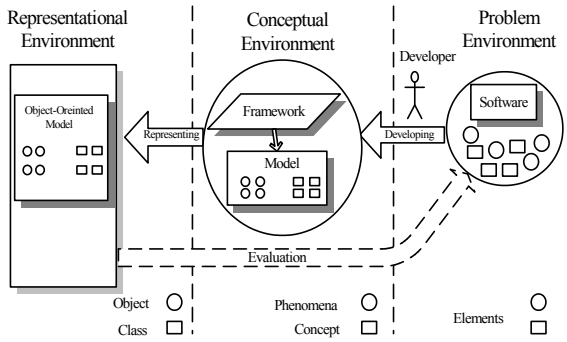

Fig 4. General Object Oriented Software Model.

A reference to the phenomenon that the object models, is an expression by which the phenomenon can be identified. Inter-object relations tell to which other objects an object is related while object properties contain some information about the phenomenon that the object models. If a phenomenon in the system is physical or otherwise tangible then an object that models this phenomenon can contain a reference to the phenomenon in the form of some expression. This expression might reference the phenomenon at hand. If a phenomenon is intangible, i. e. it is not readily perceptible, then it can be referred to by its name or position, whichever applicable. Fig 4 illustrates software model using a conceptual framework and notation based on object-oriented principles. Phenomena in the conceptual environment represent conceptualization of the software system and concepts represent abstractions (classifications) of similar phenomena. Generally it also can be stated that phenomena in the conceptual environment are associated with elements in the problem environment. They can contain references to phenomena in the conceptual model to manifest relations to other phenomena in the conceptual model, i.e. references are used to capture relations in the conceptual model to those of the considered system.

Moreover, an object is a representation of phenomena in the conceptual model, must therefore contain two references; first, it refers to objects in the representational environment and second, it refers to elements in the problem environment. A concept, which is a classification based on shared properties of a number of phenomena, embodies knowledge about the phenomena that exemplify the concept. In particular the concept must embody knowledge about the conditions for the existence of a phenomenon, i.e. which conditions must hold in the system before we can create a phenomenon in the conceptual model by making conceptualizations of the considered system.

4. Implementation and Discussion

Incorporating the object oriented modeling technique proposed in the previous section for a complete system implementation can summarized as illustrated in Fig 5. For the purpose of understandings, the model might be simply divided into several stages, namely analysis stage, design stage and implementation stage, as described below.

The analysis stage comprises of three phases; problem phase, conceptual phase and requirement phase. This stage is concerned with examination of the problem domain and usage domain producing an object oriented model that determines functional and non-functional system requirements including hardware and software components requirements. Besides, it specifies a behavior model and a framework for the object oriented design stage. During analysis stage, the user and the developer closely cooperate in order to construct the model. The process is controlled by a system definition, delimitation, modeling and evaluation with the customer verification. It is iterative process and only stops when the user and the developer agree that the descriptions are usable and express a common understanding.

The design stage is concerned with specifying the overall structure of the system, resulting mainly into object oriented system model. Besides, the developer refines the object model and introduces an architecture model in order to understand the system model. Functional and non-functional requirements are supported at this stage and system model is mapped onto logical platform. At the design level, software domain is considered including either partial or complete software descriptions. The system model is constructed from the object model. The behavior model, together with the functional requirements, supplies the information for performing the functionality to the object model. Constructive solution for how the non-functional requirements and the organization of the logical platform combined with object model and system model to produce the architecture model which starts with identification then performs classification of components into classes.

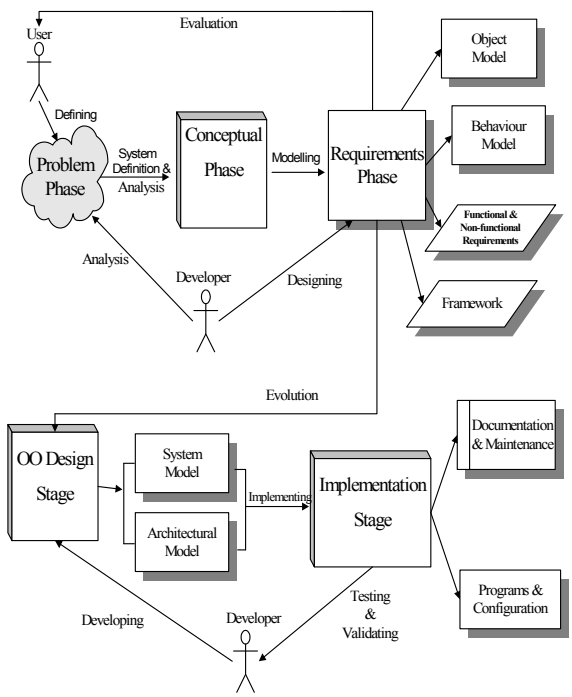

Fig 5 Implementing OO Software Model.

Implementation stage is concerned with the realization of the system model. The structure captured by the design must be implemented in certain programming language. In the implementation stage, the developer integrates the architectural model into this program during the process and transforms the system model into a refined model in the form of programs and configurations. Hence, the perspective on the platform becomes a physical perspective during the implementation. This stage involves software domain (at the programming level), physical platform and object-oriented programming language. The model of the implementation process includes an iterative cycle, where the program is constructed from the system and architecture models under the influence of relevant non-functional requirements and the physical platform. During implementation stage, programming techniques are available for the developer in order to build the programs. Hence, the structure and the interactions described by the architectural design must be implemented in a programming language, therefore it allows for system execution which is the target platform.

5. Conclusions

It is conceivable that object oriented modeling reduces system complexity by applying obstruction in terms of phenomena and concepts. The proposed model is based on "real world" entities or objects. It requires experience in object-oriented modeling and will initially be de-

rived from an analysis of the problem environment (abstract entities) towards model environment (objects and classes) through abstraction and conceptualization with the collaboration of problem domain experts (user). To reduce cost, the model is iterative and influenced by task analysis, user interface design and use-case definition. It is planned to evolve into system model and architecture model.

An important part of this model is the inclusion of analysis and evaluation activities as part of architecture design that meets stakeholder's goals or concerns. While architecture evaluation has been the focus of much prior work, the emphasis is typically on identifying candidate architectures or evaluating the completed architecture. There has been far less work on incremental or ongoing evaluation, and on architectural analysis. The proposed model reveals these to be important research topics.

This paper presents a methodology for assistive technology software development which seeks to provide a framework for requirements elicitation studies and their subsequent mapping through use-case driven object-oriented analysis to implementation using component based software architectures. Usability evaluation is integrated into the process to provide early feedback on the effectiveness of user interface components.

References

[1] Madsen O. L., Pedersen B. M., K. Nygaard K. Object-Oriented Programming in the BETA Programming Language. Addison-Wesley, Reading, Massachusetts, 1993, ISBN:0-201-62430-3.

[2] Tippell P., Oregan J., Hardy P., Lysley A. An Object-Oriented Model for Assistive Technology Software Development. Themes Valley University, 2002.

[3] Thomsen M. Domain Object Models and User Interface. Department of Computer Science, University of Aarhus, Denmark.

[4] Damma C. H., Hansen K., Thomsen M., Tyrsted M. Creative Object-Oriented Modeling: Support for Intuition, exibility, and Collaboration in Case Tools. Department of Computer Science, University of Aarhus, Denmark.

[5] Nowack P. Interacting Component, A conceptual Architecture Model. 3th European Conference on Object-Oriented Programming ECOOP Workshops (ECOOP Workshops), 1999.

[6] Jacobsen, E. E.. Concepts and Language Mechanisms in Software Modeling. PhD thesis, The Maersk Mc-Kinney Moeller Institute for Production Technology, University of Southern Denmark., 2000.

[7] Perry D. E. Wolf A. L. Foundations for the Study of Software Architecture. SIGSOFT Software Engineering Notes, 1992, 17(4): 40-52.

[8] Bosch J. Software Architecture: The Next Step. In EWSA'2004, 2004: 194-199.

[9] Hofmeister C., Kruchten P., Nord R. L., Obbink J. H., Ran A., America P. Generalizing a Model of Software Architecture Design from Five Industrial Approaches. In WICSA'2005, 2005: 77-88.

[10] Falessi D., Cantone G., Kruchten P. Do Architecture Design Methods Meet Architects' Needs?. In WICSA'2007, 2007: 5.

[11] Wojcik R., Bachmann F., Bass L., Clements P. C., Merson P., Nord R. L., Wood B. Attribute-Driven Design (ADD), Version 2.0. Technical Report CMU/SEI-2006-TR-023, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2006.

[12] Perovich D., Bastarrica M. C., Rojas C. Aspect-Oriented Model-Driven Approach to Architecture Design", 2008

[13] Spinellis B. D. Notable design patterns for domain-specific languages. Journal of Systems and Software, 2001, 56: 91–99 .

[14] Spinellis B. D., Guruprasad V. Lightweight languages as software engineering tools. Proceedings of the Conference on Domain-Specific Languages (DSL'97), USENIX, 1997: 67–76.
[15] Mernik M., Heering J., Sloane A. M., When and how to develop domain-specific languages. ACM Computing Surveys, Dec. 2005, 37 (4): 316–344.
[16] Bell J., Bellegarde F., Hook J., Kieburts R. B. Software design for reliability and reuse: a proof-of-concept demonstration. Proceedings Conference on TRI-Ada, ACM Press, 1994: 396–404.
[17] Christopher D. L., Ramming J.C. Two application languages in software production. USENIX Symposium on Very High Level Languages Proceedings, 1994: 169–187.
[18] Nowack P. Structures and Interactions - Characterizing Object-Oriented Software Architecture. PhD thesis, The Maersk Mc-Kinney Moeller Institute for Production Technology, University of Southern Denmark, 2000.
[19] Jacobsen, E. E., Kristensen B.B., Nowack P. Architecture = Abstractions over Software. Proceedings of the 32th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS PACIFIC '99), 1999: 89–99. IEEE Computer Society.