Advanced Engineering Forum *Online: 2012-09-26*
ISSN: 2234-991X, Vols. 6-7, pp 15-19
doi:10.4028/www.scientific.net/AEF.6-7.15
© 2012 Trans Tech Publications Ltd, Switzerland

Modeling Principles of the Sequence Diagram and its Application in MDA Software Paradigm

Deren Yang[1,2, a], Min Liu [1,b] and Zhaohua Gu[3,c]

[1] School of Comp. Sci. & Engi., Beifang University of Nationalities, 750021, Yinchuan, China

[2] School of Science, Ningxia Medical University, 750004, Yinchuan, China

[3] Human Resources Department, Zhoupu Hospital, 201318, Shanghai, China

[a]ydr@tom.com, [b] lmbmd2011@163.com, [c] zhaohua_76@163.com

Keywords: software model; the sequence diagram; interactive mechanism; modeling principle; MDA software paradigm

Abstract. The sequence diagram can be used to design object-oriented software; however, its modeling principles are lack of research. In this paper, its application in software design was studied, and its inherent linkage with object-oriented programming was discussed, and the interactive mechanism among its elements was explored. The modeling principles for it were proposed; and its application in Model Driven Architecture (MDA) software paradigm was analyzed; and finally the model represented with it, especially the transformation and traceability of models in MDA software paradigm were analyzed. This research is useful to optimize software modeling and to automate model transformation in MDA software paradigm.

Introduction

The sequence diagram is mainly used to identify and capture the inner mechanism of system behavior. Use-case scenarios are mapped into message sequences of object interaction step by step, and the message sequences among objects are formed. The ultimate aim is to allocate software functions to methods of objects in order to achieve the software goals. It is important to do research into its modeling mechanism, principles and roles in software modeling process.

After the related researches being reviewed in Part Ⅱ, the sequence diagram was explained, and its application and interactive mechanism were studied, and its inherent linkage with object-oriented programming was discussed in Part Ⅲ. The modeling principles for its elements in the sequence diagram were proposed in Part Ⅳ. Its application in MDA software paradigm was analyzed; and the model represented with it, especially the transformation and traceability of models in MDA software paradigm were analyzed in Part Ⅴ. And finally, conclusions and author's future research are listed in Part Ⅵ.

An Overview of Related Researches

MDA is an emerging software paradigm, and it advocates that the core of a software process is modeling and transforming models, and further divides software models into a computational independent model (CIM), a platform independent model (PIM) and a platform dependent model (PSM) [1]. There are some relations among the models, such as inheritance, transformation and traceability [2, 3]. But as for which models exist and what modeling mechanism is in software process, it is lack of academic research; neither does a uniform standard exist in industry. Unified Modeling Language (UML) can be used to model object-oriented software process and MDA software paradigm, and its sequence diagram can be used to model the dynamic process of detailed design stage and PSM in MDA software paradigm [4]. In literatures, little attention was paid to the interactive mechanism and principles in the sequence diagrams, some information about its interaction is impractical in operation [5], so some related research on it is needed.

The Sequence Diagram Explained

In object-oriented software process, the sequence diagram is widely used for visually business modeling, especially in system analysis and design. Firstly, in business analysis, the sequence diagram is used to model and optimize business process by business workers, and to clarify interactive mechanism among business objects, and to describe how business workers operate business objects and how business objects interact by sending messages. Secondly, in system analysis, the sequence diagram is used to describe, verify and enrich the logic of usage scenarios. A usage scenario is realized with the logic of system usage which may be represented with paths in use case specifications. Thirdly, in system design, the sequence diagram is used to describe interactive mechanism among objects by system designers and developers, in order to model system logic flow, and to model and verify system logic. Fourthly, it is used to describe the logic of methods called, and to explore design, such as the logic of complex operations. The sequence diagram provides visually a mechanism to call method gradually, so it can be taken as a visualizing code; and it can be used to determine the most complex class in the system. Fifthly, it is used to detect object-oriented design bottlenecks by checking approximate running time of messages through the target objects, in order to improve the design and realize load balance of system.

In fact, the sequence diagram can also be used to model PSM in MDA software paradigm [4]. However, its nature, modeling mechanism and modeling principles need further investigation.

Its Interactive Mechanism. The sequence diagram is a key technique to implement object-oriented software process, and a key tool to bridge design and coding. In form of message sequence, the sequence diagram shows interactions among actors and objects in runtime. In some degree, it is an object-level code and a basis for coding. In software design, the role of the sequence diagram is to allocate responsibilities to object based on general responsibility assignment software patterns (GRASP) [6]. A typical sequence diagram is shown in Figure 1.

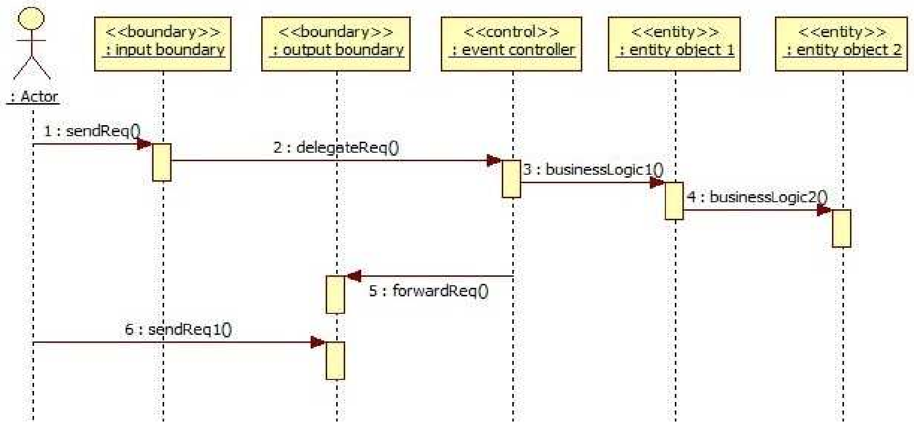
: Actor
<<boundary>> : input boundary
<<boundary>> : output boundary
<<control>> : event controller
<<entity>> : entity object 1
<<entity>> : entity object 2
1 : sendReq()
2 : delegateReq()
3 : businessLogic1()
4 : businessLogic2()
5 : forwardReq()
6 : sendReq1()

Fig. 1 A Typical Sequence Diagram

The direct source of the sequence diagram is the use case specifications and the robustness diagrams. The drawing method is as follows. Copy path description for the given use case from corresponding use case specification to begin the corresponding sequence diagram enables that text to serve as an ongoing reminder of what are to be accomplished, and paste it onto the left margin of the sequence diagram. The sequences of messages are corresponded to paths in the use case specifications. Add actor, the boundary objects and the entity objects from the corresponding robustness diagram. Each of these objects is an instance of a class. As for control objects in the robustness diagram, there are embodied to two forms in the sequence diagram. The control objects which are not connected with entity objects in robustness diagram are transformed into instances of a kind of new class called controlling classes in sequence diagram, and the others which are connected with entity objects are transformed into some methods of related classes in the sequence diagram.

Its Relations with OOP. In the sequence diagram, messages are the media of interaction among objects and actor, and are used to create target objects or call methods of target objects. The latter is a common usage. In fact, delivering messages are assigning responsibility and demanding target objects to accomplish related tasks.

In object-oriented programming, in order to achieve goals, it is necessary to call a series of methods. In order to accomplish responsibility, Objects have three choices, i.e. calling their own methods, calling other objects and delegating requests to some secondary objects. By sending a series of messages, interactions among actors and objects form a procedure for calling related methods. The interactions must comply with some other rules. For example, a caller neither directly mentions the attributes of target objects, nor assumes the operations of target objects, and can only use the operations of target objects.

Delivering messages in the sequence diagrams actually calls static methods or instance methods of classes, which embody inherent relations between a MDA model (PSM represented with sequence diagram and class diagram) and object-oriented programming.

Modeling Principles Proposed

In the sequence diagram, interactions among its elements should follow certain principles. The modeling principles which include interactive rules and prohibitions are proposed as in Table 1.

Interactive rules: an actor can only interact with the boundary objects (BO), and the boundary objects can interact with the instances of classes (control classes or entity classes), the boundary objects are the intermediary of actor and the instances of classes; the instances of control classes (CC) can interact with the instances of entity classes, and the instances of entity classes (EC) can interact with each other.

Interactive prohibitions: An actor can not directly interact with the instances of control classes or entity classes, the boundary objects can not directly interact with each other, the instances of control class can not directly interact with each other.

Table 1: Modeling Principles for the Sequence Diagram

	Actor	BO	Instance of CC	Instance of EC
Actor	×	√	×	×
BO	√	×	√	√
Instance of CC	×	√	×	√
Instance of EC	×	√	√	√

√: interact directly; ×: interact indirectly

In the robustness diagram, interactions among objects follow "noun-verb-noun" or “nouns-verbs-verb-noun” rule; while in sequence diagram, interactions among objects only partly follow "noun-noun” rule, as shown in Table 1.

The reason is that in the sequence diagram the interactive mechanism among actor, the boundary objects and the instances of classes are messages; while in the robustness diagram the controllers are interactive intermediaries between entity objects and boundary objects.

Application in MDA Software Paradigm

A Modeling Process for MDA. Referring to some traditional software process paradigms, such as life-cycle process paradigm, the models for MDA are built at different stages in a software process [4]:

- A CIM is built at the stage of describing requirements phase with the use case diagram;
- Based on the CIM, a PIM is built at the stage of analysis phase with use case specification and the robustness diagram;
- Based on the PIM, a PSM is built at the stage of design phase with the state machine diagram, the class diagram and the sequence diagram;

- Based on the PSM, an IM is built at the stage of implementation phase with specific programming languages, such as Java or C#.

In the sequential modeling process, the immediate former is the basis of the latter. The relations among models, such as inheritance, transformation, are embodied. This is conducive to reuse model and realize the interoperability among systems.

Table 2: The relations among models, domains and languages

	Sub-processes	Domains		UML diagrams
CIM	Requirement describing	System		Use Case diagram, UCS*
PIM	System analyzing	Solution	Analysis	Robustness diagram
PSM	Solution designing		Design	State machine diagram, Class diagram, Sequence diagram
IM	Implementing	Implementation		Java or C#

* UCS: Use Case Specification

This modeling process also applies to other heavyweight software process models, such as Rational Unified Process (RUP) paradigm.

Through the application of MDA, the role and objective of object-oriented analysis, and those of object-oriented design are optimized, so reusing previous design models, such as PIM, are guaranteed.

The corresponding relations among models of MDA software paradigm, domains, stages and languages are shown in Table 2.

The Sequence Diagram and the Models of MDA. According to the modeling process for MDA software paradigm, **t**he sequence diagram can be used to model PSM. Based on PIM represented with the robustness diagram and according to GRASP [6], the control objects which are directly connected with entity objects in robustness diagram are mapped to one or more messages in the sequence diagram in order to allocate operations to objects to achieve the desired software behaviors. Based on the state machine diagram and the sequence diagram, the class diagram is generated, so PSM, which is represented with the sequence diagram, the state machine diagrams and the class diagrams, is modeled. The PSM is the basis for coding.

In a software process, the model represented with the sequence diagram has some important properties, such as inheritance, traceability, and transformation.

The inheritance and traceability of sequence diagram are embodied in its association with the robustness diagram. Before drawing the sequence diagram, it is necessary to know which objects exist in use case specification. After users' interaction, which functions will be performed by system? This kind of information can be obtained through the robustness diagram. The latter is the source of the actor and objects in the sequence diagram, and the sequence diagram can be traced back to the use case specification and the robustness diagram. For example, the sequence diagram shown in Fig. 1 is based on and can be traced back to the robustness diagram shown in Fig. 2

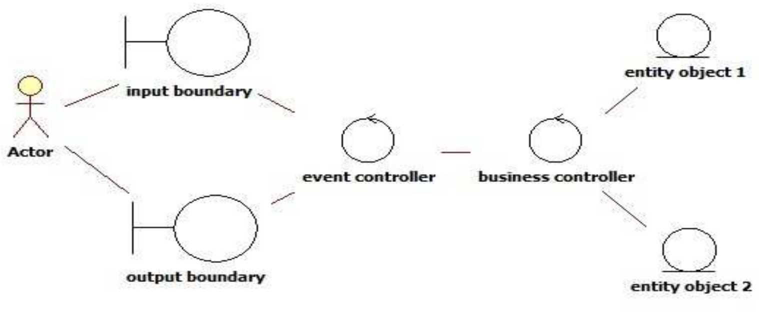

Fig. 2 A Typical Robustness Diagram

The transformation of the sequence diagram is reflected in its relations with the state machine diagram and the class diagram. For objects whose states change frequently or have multi-state value, the state machine diagram can be used to find their methods. Based on GRASP [6], the sequence diagram can be used to allocate methods for classes; in fact it is the basis for completing the class diagram.

Summary

The sequence diagram has some wide usages, and it is mainly used to describe graphically the interactive sequences among actors and objects in systems. In MDA software paradigm, PSM described with the sequence diagram has traceability, and its objects can be traced back to object prototypes in the robustness diagram, and the message sequences can be traced back to paths in the use case specification and the robustness diagram. Based on GRASP [6], methods are allocated to instances of classes in the sequence diagrams. And the sequence diagrams can be further transformed into the class diagrams. The automatical mechanism of transformation and traceability needs further study, and Graphical Editing Framework (GEF) can be used to implement it [7].

Acknowledgment

The research is supported by National Innovation Project (11C26216406397), Natural Science Foundation of Ningxia (NZ10103), Chunhui Plan (Z2011052), Project of Colleges and Universities of Ningxia (NGY2011044), Special Talents Project of Ningxia Medical University (XT200913), and Project of Pudong Health Bureau (PW2012A-33).

References

[1] O. Pastor, S. España, J.P.N. Aquino, Model-Driven Development: piecing together the MDA jigsaw, Informatik-Spektrum, Special issue on Modelling, 31(2008)394-407.

[2] N.Anquetil, U.Kulesza, R. Mitschke, etc, A Model-Driven Traceability Framework for Software Product Lines, Software and Systems Modeling, 9(2009)427-451.

[3] Deren Yang, Mei Xue, On Software Process Paradigm and its constraint Mechanism, Proceedings of 2nd International Conference on Software Engineering and Service Sciences, IEEE Press, (2011)842-845.

[4] Deren Yang, Min Liu, Shengguo Wang, Object-Oriented Methodology Meets MDA Software Paradigm, Proceedings of the 3nd International Conference on Software Engineering and Service Science, IEEE Press, (2012)208-211.

[5] S.W. Ambler, The Elements of UML 2.0 Style, Cambridge University Press, 2005.

[6] C. Larman, Applying UML and Patterns, Prentice Hall Press, Third Edition, 2004.

[7] Deren Yang, Min Zhou, Fen Chen, et al, Abstract Mechanisms of GEF and Techniques for GEF based Graphical Editors, LNEE 125, Advance in Computer Science and Information Engineering, Springer Press,2 (2012)281-287.