MEDINFO 2001
V. Patel et al. (Eds)
Amsterdam: IOS Press
© 2001 IMIA. All rights reserved

An Exchange Format for Use-cases of Hospital Information Systems

Gou Masuda[a], Norihiro Sakamoto[b], Rumi Sakai[b], Ryuichi Yamamoto[a]

[a] *Division of Medical Informatics, Osaka Medical College, Osaka, Japan*
[b] *Department of Medical Informatics, Kyushu University Hospital, Fukuoka, Japan*

Abstract

Object-oriented software development is a powerful methodology for development of large hospital information systems. We think use-case driven approach is particularly useful for the development. In the use-cases driven approach, use-cases are documented at the first stage in the software development process and they are used through the whole steps in a variety of ways. Therefore, it is important to exchange and share the use-cases and make effective use of them through the overall lifecycle of a development process. In this paper, we propose a method of sharing and exchanging use-case models between applications, developers, and projects. We design an XML based exchange format for use-cases. We then discuss an application of the exchange format to support several software development activities. We preliminarily implemented a support system for object-oriented analysis based on the exchange format. The result shows that using the structural and semantic information in the exchange format enables the support system to assist the object-oriented analysis successfully.

Keywords:

Use-case, Software development process, Object-oriented system, XML

Introduction

Hospital information systems have become increasingly large, diverse and complex in recent years. Development of the systems therefore requires a great deal of effort, time and costs. Object-oriented technology offers one solution to this problem. It provides a number of powerful techniques such as abstraction, inheritance, polymorphism and encapsulation. The use of those techniques makes it possible to construct the systems with less cost and effort. By improving the modularity and readability of the systems using those techniques, it helps developers to understand the structure of the systems and to maintain, modify and reuse them. Several hospital information systems have being developed using object-oriented technology[1][2]. A number of methodologies have proposed for object-oriented software development. In particular, use-case analysis[3] gradually spreads in developing hospital information systems for capturing user requirements. For instance, the message development framework[4] for the HL7 Version 3 adopts use-case analysis to develop a series of messages exchanged between hospital information systems.

In general, use-cases are documented at the first step in the development process and they are used through the whole steps in a variety of ways. Therefore, it is important to exchange and share the use-cases and make effective use of them through the overall lifecycle of a development process. Moreover, exchanging them between different development projects is one of the important requirements for development of hospital information systems. By the social, economical, and regional demands, healthcare institutions take the wide variety of forms. However, requirements to individual element of work process such as a surgical operation and a radiological examination is common to those institutions. Consequently, we can share the each part of healthcare information systems between different development projects. Therefore, sharing and reusing use-cases between the projects may reduce the cost throughout the whole medical society. However, it is difficult to meet those requirements because use-cases are documented in a variety of formats according to their applications. To remedy this, we propose an exchange format for use-case based on eXtensible Markup Language (XML). We call this format XUC (eXtended Use-Case format). We also describe an application of XUC to support software development process.

Materials and Methods

A Use-Case Driven Software Development Process

Use-case driven software development process introduced by Jacobson in OOSE (Object-Oriented Software Engineering)[3] is a powerful methodology for requirement analysis in the development of software systems. In OOSE, a use-case is defined as an interaction between the system and an "actor" that causes the system to fulfill a responsibility and, as a result, to produce a product of value for the actor. Actor is defined as people or other computer systems that interact with the systems under discussion. A collection of use-cases and actors is defined as a use-case model. Figure 1 shows an example of use-case model represented by Unified Modeling Language (UML).

In general, a series of system development process includes requirement analysis, design, implementation, and testing phases. Each phase produces and maintains its corresponding model. Development of a system is therefore considered as a changing process of a series of models that is produced in each development phase. In the use-case driven software development process, use-case models are produced in the beginning of the process, and they are used by other models that are produced in the following phases. For example, an analysis model is produced from the functional specifications described in use-case models. In testing phase, the use-case models are used to confirm whether the system satisfies the original requirements. Use-case models have thus important roles in the overall lifecycle of development process. It is therefore important to exchange the use-cases between developers and applications through the development process effectively.

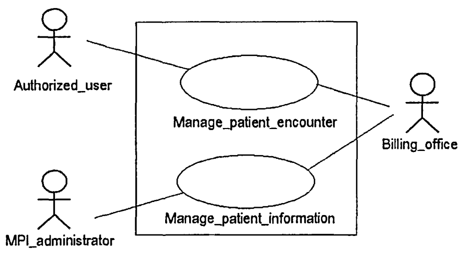


Figure 1 – An example of use-case model.

An Exchange Format for Use-case

In this section, we describe an exchange format for use-cases. First, we define an object model for use-case. To define this, we review the Basic Use Case Template[5][6], which provides a systematic and consistent format for describing use-cases, from the viewpoint of exchanging the software design. Next, we design an XML based exchange format for the object model. In this exchange format, the relationships between objects and structure of classes in the object model are represented as XML elements. Then we add other related information on exchanging electronic documents such as author of the document and revision date to the format. We call this exchange format XUC and we refer to the document written by the format as XUC document. XUC defines the following XML elements to describe a use-case model. Complete XML document type definition (DTD) is presented in Appendix A.

- Use-case name: The name straightforwardly indicates the goal of a use-case. A verb phrase is better.
- Scope: The systems or subsystems that are considered as black box in a use-case.
- Level: The degree of details of a use-case, that is, Summary, Primary Task, and Subfucntion.
- Goal in context: A detailed statement of the goal of a use-case.
- Primary actor: The external actor that triggers the use-case.
- Secondary actor: All the actors that provide services to the system in order to achieve the goal of the use-case.
- Preconditions: The conditions with which systems and actors are required to be satisfied before the use-case is triggered.
- Success end condition: The state of the system after successful completion of a use-case.
- Failed end condition: The state of the system after the goal is not achieved.
- Trigger: The event that triggers a use-case.
- Main scenario: A series of steps of a scenario from trigger to goal. Each step is called "Interaction." It consists the following two sub elements.
  - Step: The order that the interaction is carried out.
  - Action: The description of an interaction between an actor and system. The subject of the interaction should be stated clearly.
- Alternative scenario: The subsidiary description of a use-case such as exception handling.
  - Condition: The description on the conditions for branch.
  - Step: The order that the interaction is carried out.
  - Action: The description of an interaction between an actor and system.
  - Related use-case: A reference to sub use-cases if the action can be separated as a use-case independently.
- Variation: The choices in the scenario as a temporal branch. It consists the following elements.
  - Note: The description of situation when the choices occur.
  - Variation item: The content of each choice.
- Author: The author of a use-case.
- Revision date: The revision date of a use-case.
- Super ordinates: The name of use-cases that includes this use-case.

- Sub ordinates: The name of use-cases that this use-case includes.
- Note: The description that cannot be included any other elements. For example, issues or notices.

Describing use-cases as XUC documents enables developers to exchange the use-cases between applications, developers thorough the development process. Structural and semantic information in the XML document makes it possible to utilize the use-cases in a variety of applications.

Related Works

The XML Metadata Interchange Format (XMI)[7] is an exchange format for the models in UML such as Class diagram and Object diagram. It allows developers to exchange software assets in UML throughout their applications. XMI provides a way to exchange Use case diagram in UML. However, detailed descriptions of use-cases are not represented by using XMI. The reason is that Use-case diagram in UML is not designed for documenting a use-case rather for the relationship between use-cases and actors.

The Basic Use Case Template[5][6] proposed by Cockburn is designed for providing a systematic and consistent format for describing use-cases. However, it uses a simple text format. We think it is therefore difficult to analyze and utilize the document in a variety of applications.

On the other hands, XUC is designed to provide developers a model for representing detailed descriptions of use-cases and their exchange format. We think XUC and XMI are applicable complementally depending on the detailed level of use-cases. Therefore, we also provide a way to interchange between XUC and XMI in this study. The interchangeability is discussed in a later section.

Results and Discussion

An Application to Support the Software Development Process

A Support System for Object-Oriented Analysis

In order to evaluate the usefulness of XUC, we apply it to support analysis and design activities in the software development process introduced by Jacobson's use-case driven development approach. The purpose of object-oriented analysis is to understand the system and in particular, the functional requirements of the system in discussion. It consists of the following five primitive activities, and those activities are conducted iteratively from abstract phase to concrete one [3].

- Finding the objects

  Objects can be found as naturally occurring entities in the application domain. They usually appear in nominal words in the application domain. The purpose of this activity is identifying indispensable objects that keep on being important through the overall lifecycle of system.

- Organizing the objects

  This activity includes finding the similarity between objects. Class hierarchy is defined based on the similarity in general. It also includes finding how objects work together and what object contains others.

- Describing how the objects interact

  The purpose of this activity is to specify how an object is incorporated in the system. Describing use-cases that include the object is useful to accomplish the purpose.

- Defining the operations of the objects

  Making clear the interface of an object leads to defining methods of the object. Complex operations of an object imply that the object can be separated.

- Defining the objects internally

  This activity includes defining attributes of objects. The inside of object is described in detail by using those attributes.

All the activities are based on the use-case models that are documented in the beginning of the development process. Therefore exchanging the use-cases between developers and applications allows developers to make use of the use-case models in the overall development lifecycle effectively. For this purpose, we build a support system for object-oriented analysis as an application of XUC. Figure 2 shows the overview of our support system. It consists of three tools, namely, Use-case Editor, Object-Oriented Analysis (OOA) Support Tool, and Class Diagram Editor.

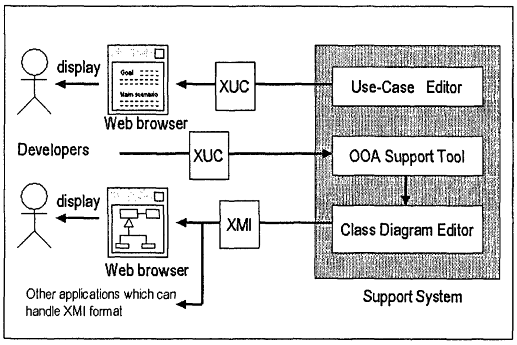
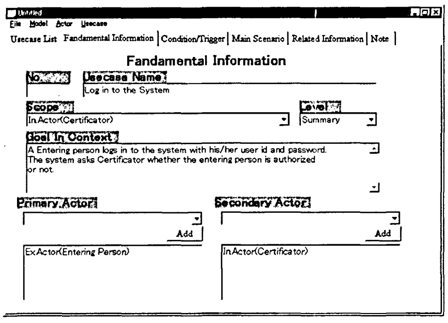


Figure 2 – Overview of our support system.

Use-case Editor

Use-case Editor is used for documenting use-case models as XUC format. It has an XML processor in order to interpret XML documents. Figure 3 shows an example use of the Use-case Editor. It enables developers to define actors and scenarios in a use-case model and construct it as a XUC document. The constructed XUC documents can be displayed easily using eXtensible Stylesheet Language (XSL). Users can browse the XUC documents by using common WWW browsers such as Microsoft Internet Explorer 5^{TM}. No specific applications or tools are required for the browsing the XUC documents.



Fandamental Information

No.

Usecase Name: Log in to the System

Scope: InActor(Certificator)

Level: Summary

Goal In Context: A Entering person logs in to the system with his/her user id and password. The system asks Certificator whether the entering person is authorized or not.

Primary Actor: ExActor(Entering Person)

Secondary Actor: InActor(Certificator)

Figure 3 – Use-case Editor for XUC documents.

OOA Support Tool

This tool supports the first three activities in object-oriented analysis.

- Finding the objects

  The first step in object-oriented analysis is finding objects. We pay particularly attention to nominal words in use-cases and the place where the words appear. Thus, our support tool extracts those words from the use-cases and presents them to users as candidates of object. Users can define objects in their system using the candidates.

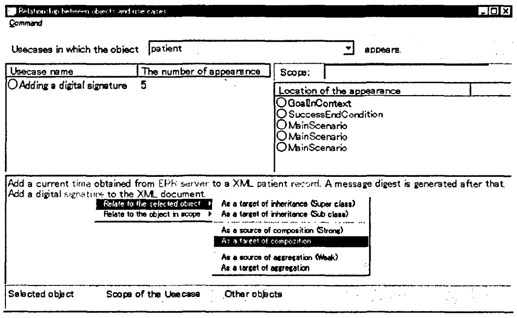
  As regards the place of word appearance, we found that the words that appear in "Internal Actor" section in a use-case model are possible to be objects with high probability. Therefore, the tool automatically extracts those words as candidates of objects. On the other hand, the words that appear in "External Actor" section have no probability of being object. The tool therefore removes those words from the candidates. Use of the structural and semantic information in XUC makes it possible to extract the candidate automatically.

- Organizing the objects and their interactions

  In the case of defining classes, it is necessary for developers to recognize the relationships among classes such as inheritance, generalization and aggregation. The information on the relationships between objects in a use-case model is helpful in this activity. However, those relationships are often described implicitly in the use-case models. We thus consider the use of structural and semantic information in XUC documents to extract those relationships. We apply the following two methods to extract relationships between objects from XUC documents.

  - Using structural information: Extract objects that exist in the same XML element in which a target object appears.
  - Using semantic information: Extract objects that exist in the same scenario in which a target object appears.

  The support tool presents the candidates of the relationship between objects using both methods. Users can define appropriate relationship from the candidates. An example use of OOA Support Tool appears in Figure 4.


Relationship between objects and use cases
Command
Relate to the selected object
Relate to the object in scope
As a target of inheritance (Super class)
As a target of inheritance (Sub class)
As a source of composition (Strong)
As a target of composition
As a source of aggregation (Weak)
As a target of aggregation
Selected object
Scope of the Usecase
Other objects

Usecases in which the object patient appears.

Usecase name	The number of appearance
○Adding a digital signature	5

Scope:

Location of the appearance
○GoalInContext
○SuccessEndCondition
○MainScenario
○MainScenario
○MainScenario

Add a current time obtained from EPR server to a XML patient record. A message digest is generated after that. Add a digital signature to the XML document.

Figure 4 – OOA Support Tool.

Class Diagram Editor

Class Diagram Editor provides a way to define inside of objects, namely, attributes of objects and relationship between them. In addition, it provides a way to construct a framework of the classes automatically using the results from OOA Support Tool.

The classes defined by our support system can be used in the following phases of development process such as implementation and testing. One possible way to exchange such information through the whole phases is using XMI. Therefore, our support system provides a converting method from XUC to XMI. This makes it possible to share the information model from the analysis phase to the later phases seamlessly. In this study, we consider only Class diagram in UML as a target of the conversion since it is used most frequently than other diagrams.

An Evaluation

We applied our support tool to a development of a piece of software for securely storing and transferring electronic patient records. Seventeen use-cases documented via our support tool were used to assist finding objects and defining their interfaces. In the activity of finding objects, a developer actually used 31 objects from 187 candidates presented by the support tool. We also confirmed that using OOA support tool enables the developer to easily understand the entire framework of the software.

Conclusion

We have proposed XUC, an exchange format for use-cases. We have also described our support system as an application of the XUC. The result shows that using the structural and semantic information in the exchange format enables the support system to assist the object-oriented analysis successfully.

By using XUC, we can exchange use-cases not only within a development process but also between different projects. Moreover, we suppose to construct a use-case repository for developing various hospital information systems.

Currently, we are developing a large hospital information system at Osaka Medical College. In the development, a large number of use-cases are documented for requirement analysis of the system. We plan to apply XUC to the development and discuss its availability as part of our future work.

Acknowledgments

We would like to thank Kousuke Tanaka for his implementation of our support system.

References

[1] Egyhazy CJ, Eyestone SM, Martino J, and Hodgson CL. Object-oriented analysis and design: A methodology for modeling the computer-based patient record. Topics in Health Information Management; Frederick 1998.

[2] Wang C, and Ohe K. A CORBA-Based Object Framework with Patient Identification Translation and Dynamic Linking. Methods for exchanging patient data. Methods Inf Med 1999; 38:56-65.

[3] Jacobson I, Christerson M, Jonsson P, and Övergaard G. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley Publishing Company, 1992.

[4] Health Level Seven, Inc. HL7 Version 3 Message Development Framework Version 3.3. 1999. Available at http://www.hl7.org/

[5] Cockburn A. Basic Use Case Template. HaT technical report TR96.03a. Humans and Technology, 1996.

[6] Cockburn A. Structuring Use Cases with Goals. Journal of Object-Oriented Programming. Sep/Oct, 1997. pp. 35-40, and Nov/Dec, 1997. pp. 56-62.

[7] Object Management Group. XML Metadata Interchange Format (XMI), version 1.1. 2000. Available at http://www.omg.org/technology/documents/formal/xml_metadata_interchange.htm

Appendix

We give the complete XML DTD for XUC in Table A1.

Table A1- DTD for XUC

```
<!ELEMENT XUC (UsecaseModel*)>
<!ELEMENT UsecaseModel
  (UsecaseModelName, Actors, Usecases)>
<!ATTLIST UsecaseModel id ID #IMPLIED>
<!ELEMENT UsecaseModelName (#PCDATA)>
<!ELEMENT Actors (ExternalActor*, InternalActor*)>
<!ELEMENT Usecases (Usecase*)>
<!ELEMENT ExternalActor
  (ActorName, ActorCharacterization)?>
<!ATTLIST ExternalActor
  id ID #IMPLIED idref IDREF #IMPLIED>
<!ELEMENT InternalActor
  (ActorName, ActorCharacterization)?>
<!ATTLIST InternalActor
  id ID #IMPLIED idref IDREF #IMPLIED>
<!ELEMENT ActorName (#PCDATA)>
<!ELEMENT ActorCharacterization (#PCDATA)>
<!ELEMENT Usecase
  (UsecaseName, GoalInContext, Scope, Level,
  PrimaryActors, SecondaryActors, PreConditions,
  SuccessEndCondition, FailedEndCondition,
  Trigger, MainScenario, Authors, RevisionDates,
  UsecaseNotes, SuperOrdinates, SubOrdinates)?>
<!ATTLIST Usecase
  id ID #IMPLIED idref IDREF #IMPLIED>
<!ELEMENT UsecaseName (#PCDATA)>
<!ELEMENT GoalInContext (#PCDATA)>
<!ELEMENT Scope (InternalActor?)>
<!ELEMENT Level EMPTY>
<!ATTLIST Level value
  (Summary | PrimaryTask | Subfunction) "Summary">
<!ELEMENT PrimaryActors
  ((ExternalActor | InternalActor)*)>
<!ELEMENT SecondaryActors
  ((ExternalActor | InternalActor)*)>
<!ELEMENT PreConditions (#PCDATA)>
<!ELEMENT SuccessEndCondition (#PCDATA)>
<!ELEMENT FailedEndCondition (#PCDATA)>
<!ELEMENT Trigger (#PCDATA)>
<!ELEMENT MainScenario (MainInteraction*)>
<!ELEMENT MainInteraction
  (Action, Extensions, SubVariations)>
<!ATTLIST MainInteraction step NMTOKEN #REQUIRED>
<!ELEMENT Action (#PCDATA)>
<!ELEMENT Extensions (Extension*)>
<!ELEMENT Extension
  (Condition,AlternativeScenario,ReferedUsecase?)>
<!ELEMENT Condition (#PCDATA)>
<!ELEMENT AlternativeScenario
  (AlternativeInteraction*)>
<!ELEMENT AlternativeInteraction (Action)>
<!ATTLIST AlternativeInteraction
  step NMTOKEN #REQUIRED>
<!ELEMENT ReferedUsecases (Usecase)>
<!ELEMENT SubVariations (SubVariation?)>
<!ELEMENT SubVariation
  (VariationNote, VariationItem+)>
<!ELEMENT VariationNote (#PCDATA)>
<!ELEMENT VariationItem (#PCDATA)>
<!ELEMENT Authors (Author*)>
<!ELEMENT RevisionDates (RevisionDate*)>
<!ELEMENT UsecaseNotes (UsecaseNote*)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT RevisionDate EMPTY>
<!ATTLIST RevisionDate year NMTOKEN #REQUIRED
  month NMTOKEN #REQUIRED day NMTOKEN #REQUIRED>
<!ELEMENT UsecaseNote (#PCDATA)>
<!ELEMENT SuperOrdinates (Usecase*)>
<!ELEMENT SubOrdinates (Usecase*)>
```

Address for correspondence

Gou Masuda, Division of Medical Informatics, Osaka Medical College, 2-7 Daigakucho, Takatsuki, Osaka 569-8686, Japan masuda@poh.osaka-med.ac.jp