A Design and Implementation of Group Decision Support System using Object-Oriented Modeling Technique *

Soung Hie Kim, Sung Sik Cho
Dept. of Management Engineering, Graduate School of Management, KAIST, Seoul, Korea
Sun Uk Kim
Dept. of Industrial Engineering, DanKuk University, CheonAn, Korea
Hung Kook Park
Dept. of Telecommunication Systems Management, SangMyung University, Seoul, Korea

Abstract

Object-Oriented Modeling Technique (OMT) [1] promotes better understanding of requirements, cleaner designs, and more maintainable systems. A development of Group Decision Support System (GDSS) needs this advantage of OMT. GDSS designed through OMT can be more maintainable and evolve easily, which satisfies both developers and users. OMT proposes 3 modelings, object modeling, dynamic modeling, and functional modeling. This paper illustrates a design of GDSS using these 3 modelings. By exploiting the object-oriented paradigm, this design results in a highly system-independent design. And this paper shows some implementation issues including a tip of C++ code.

1.Introduction

Group Decision Support System (GDSS) is composed of a lot of technologies such as, networking, DBMS basically, and decision supporting tools such as idea generator, idea organizer[2,9]. These technologies evolve respectively without considering other ones and thus making the system more complex and less maintainable. Therefore we need a new methodology to deal with this problem. By using OMT we can consider all components of GDSS as object which are more managable and flexible.

Since the mid-1980's there has been growing interest among system researchers in an alternative approach to systems development called object-oriented analysis and design (OOAD)[10]. The advantage of this approach can be summarized as: better user/analyst communication and thus easier modeling; reusability of code; improved flexibility; increased productivity and better reliability.

OMT is one of techniques that realized the concept of OOAD. OMT presents an object-oriented approach to software development based on modeling objects from the real world and then using the model to build a language-independent design organized around those objects. Object-oriented models are useful for understanding problems and designing programs and databases. A design made with OMT has a lot of flexibility and maintainability.

In this paper, we propose a design of GDSS using OMT and its implementation. Developers of GDSS can use this design or objects for developing their own systems or for combining their design or objects to ours.

The rest of the paper is organized as follows. Section 2 presents an overview of OMT with its 3 models. Section 3 outlines the analysis and design of GDSS. On section 4 we discuss a few implementation issues. Finally, section 5 concludes the paper by describing future works.

* This paper was supported by Korea Science and Engineering Foundation(KOSEF# 93-0100-12-01-3).

2. Object-oriented Modeling Technique (OMT) Overview

OMT presents a methodology for object-oriented development and a graphical notation for representing object-oriented concepts. The methodology consists of building a model of an application domain and then adding implementation details to it during design of a system. The methodology has following stages:

Analysis : Starting from a statement of the problem, the analyst builds a model of the real-world situation showing its important properties. The analysis model is a concise, precise abstraction of *what* the desired system must do, not *how* it will be done.

System Design : During system design, the target system is organized into subsystems based on both the analysis structure and the proposed architecture.

Object Design : The object designer builds a design model based on the analysis model but containing implementation details. The focus of object design is the data structure and algorithms needed to implement each class.

Implementation : The object classes and relationships developed during object design are finally translated into a particular programming language, database, or hardware implementation.

The OMT methodology uses three kinds of models to describe a system:

Object model describes the static structure of the objects in a system and their relationships. The object model contains object diagrams. *Object diagram* is a graph whose nodes are object classes and whose arcs are relationships among classes.

Dynamic model describes the aspects of a system that change over time. The dynamic model is used to specify and implement the control aspects of a system. The dynamic model contains state diagrams. A *state diagram* is a graph whose nodes are states and whose arcs are transitions between states caused by events.

Functional model describes the data value transformations within a system. The functional model contains data flow diagram.

The three models are orthogonal parts of the description of a complete system and are cross-linked. The object model is fundamental, however, because it is necessary to describe *what* is changing or transforming before describing *when* or *how* it changes.

Object-oriented development inverts the previous function-oriented methodology, as exemplified by the methodologies of Yourdon [3] and DeMarco [4]. In these methodologies, primary emphasis is placed on specifying and decomposing system functionality. By contrast, the object-oriented approach focuses first on identifying objects from the application domain, then fitting procedures around them. Object-oriented software holds up better as requirements evolve, because it is based on the underlying framework of the application domain itself, rather than the ad-hoc functional requirements of a single problem.

3. Anaysis and Design

3.1 Analysis

At first we found out the objects from the real world. We could find the objects from the *problem statement*[1] which described the real meeting. The problem statement contains *what*, *when* and *how*. In object modeling step, we focus on only *what*. The objects found are *Meeting*, *Leader*, *Participant*, *Agenda*, and *Activity*. Fig. 1 shows the simple object model.

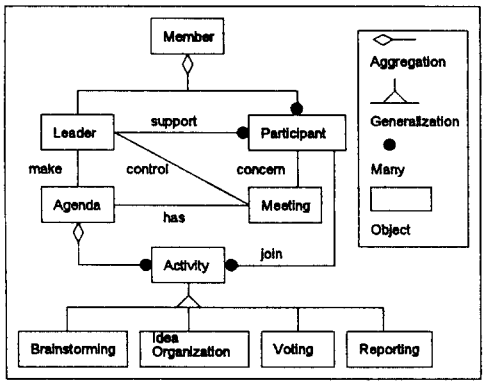

Fig. 1. *Object model of meeting*

Leader makes the agenda, controls the meeting, and supports the participants. Participants joins some activities such as brainstorming, voting or so. There can be various activities, but here we

included only 4 major activities. By the paradigm of object-oriented concept, new activities can be added easily. At the design step, we add more system dependent objects and methods or operations that correspond to *when* and *how* on the basis of this object model.

And next, we built the dynamic model and functional model. These models complement the object model because only the object model cannot describe the whole system. These models became the bases for design.

3.2 Design

In this step, we revised the object model derived from the analysis step. We added some system dependent objects such as database and removed unnecessary objects. Fig. 2 shows the object design of overall system.

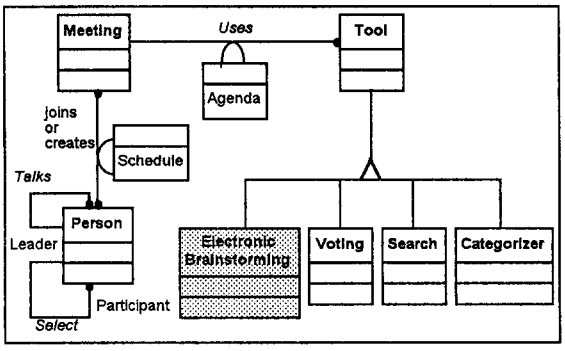

Fig. 2. *Object design of GDSS*

A person can be a leader or a simple participant. A leader creates a meeting and a participant joins the meeting. *Agenda* and *Schedule* are link attributes. We have *Tool* object instead of *Activity* object in the analysis step. It's because we are designing an executable system in this step, we consider an activity as a tool or a utility that users use. *Tool* object will be a menu item of the implemented system. The objects *Person*, *Sechedule*, *Meeting* and *Agenda* will be a database. These objects contains data rather than operations. But the objects that belong to *Tool* object - *Electronic Brainstorming*, *Voting*, *Search*, *Categorizer* - contain operations rather than data. These objects use network and database objects. In Fig. 1, we omitted network object and database object. We will discuss these objects on the next section.

Fig. 3 shows the *Electronic Brainstorming* object in detail. *Electronic Brainstorming* object is composed of two objects *Private* and *Public*. These objects correspond to private screen and public screen respectively. Through *Private* object participants enter their ideas. And *Public* object shows the ideas that all participants entered. Each of these two objects connected to *Idea Server* object. *Idea Server* object manages the *idea* object. In this design *Private* and *Public* don't have to know how to manage the database. They just have to know where the *Idea Server* is. Three objects *Person*, *Idea*, *Meeting* are actually databases. If these objects are implemented in certain vendor's database, only the *Idea Server* object must be changed, not influencing other objects. That is one of the advantages that object-oriented concepts have.

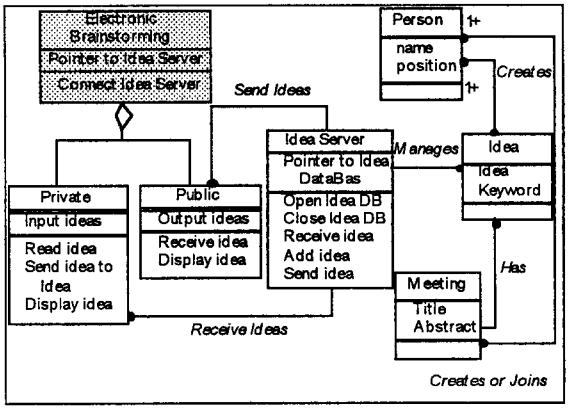

Fig. 3. *Object design of Electronic Brainstorming*

In this object design the objects *Idea Server*, *Person*, *Idea*, *Meeting* are going to be located on the server and other objects are going to be located on the client. Thus this design can be implemented as a Client/Server system.

4. Implementation Issues

In this section we discuss how we implemented the design. There's no need to use only object-oriented programming (OOP) languages but we used C++[5] because it's very easy to implement an object with OOP. And also one of our objectives was to provide a set of objects

for code reusability not only for design reusability. If some developers want to use our design they also can use our implemented objects.

The objects that are actually databases are implemented with MS-SQLServer[6]. If someone wants to implement with other vendor's DBMS, he/she must change the database related objects. But actually most of vendors provide database related objects, thus all we have to do is to replace the current object with new one.

Client objects typically handle network related events. For example, the methods *Send idea to Server* in *Private* object must know how to handle the network. And there are various network protocols in real world. But to solve this problem, we don't have to implement all kinds of network protocols whenever we port our system to new flatform. We can solve this problem by using objects that was built before by others. There are many class libraries that help building protocol-independent network objects[7].

Here's a tip of code that implemented *Electronic Brainstorming* object.

```
class CBrainstorming
{
public:
   CBrainstorming();                    // constructor
// attributes
   CString ServerName;      // name of idea server
   LOGINREC *login;                     // login data
   CString m_PrvString; // buffer for private screen
   CString m_PubString; // buffer for public screen
// methods
   virtual CString ReadIdea();
   virtual void SendIdea();
   virtual CString ReceiveIdea();
   virtual void DisplayIdea(CString *Idea);
            ......
};
```

5. Conclusion

In this paper we proposed a design of GDSS through applying OMT. OMT helped us analyze and design GDSS from the point of object-oriented concept. By exploiting the object-oriented paradigm, our design confines database and network constraints in a few easily upgradable objects, thus resulting in a highly system-independent design. And by the object-oriented paradigm this design and its implementation can be easily reused. Our design is not perfect but anyone who wants to develop a GDSS can extend our design and implementation, that is one of our objectives.

But our design has a limitation. The problem statement from which we started analysis are not complete. There are various kinds of meetings such as reporting , decision making, generating new ideas, and planning project[8]. But our problem statement described only idea generation, thus our design can not cover various kinds of meetings. So we need to extend our design to cover various meetings in future.

References

[1] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[2] Morrison, J., Sheng, O., "Communication technologies and collaboration systems : Common domains, prolbems and solutions," *Information & Management* 23, 1992. pp.13-31.

[3] Yourdon, E., *Modern Structured Analysis*, Yourdon Press, 1989.

[4] DeMarco, T., *Structured Analysis and System Specification*, Prentice Hall, 1979.

[5] Stroustrup, B., *The C++ Programming Language*, 2nd Ed, Addison Wesley, 1992.

[6] Davis, R., *Windows Network Programming*, Addison Wesley, 1993.

[7] Nath, A., *The Guide to SQLServer*, Addison Wesley, 2nd Edition, 1994.

[8] Park, H., Olfman, and Satzinger, "Attitude toward and preference for group work," working paper in Claremont Graduate School, 1995.

[9] DeSanctis, G., and B. Gallupe. "A Foundation for the Study of Group Decision Support Systems," *Management Science*, Vol. 33, No. 5 (1987):589-609.

[10] Graham, I., *Object Oreinted Methods*, 2nd Edition, Addison Wesley, 1993.