韓國IT서비스學會誌
第5卷 第2號
2006年 8月

협업적 소프트웨어 개발 관리 시스템의 설계 및 구현

한관희* · 송희석**

Design and Implementation of a Collaborative Software Development Management System

Kwan Hee Han* · Hee Seok Song**

■ Abstract ■

Since software development team members have been more geographically spread due to the globalization of business and Internet technologies, the management of deliverables and communication efforts for developing high-quality software products on time is becoming more complicated. Among the functional requirements for collaborative software development management, the manipulation of shared information objects is essential for the collaborative work among distributed development team members. This paper proposes an integrated information object management framework comprised of a so-called BOC (Bill Of Class) scheme and a standardized software part dictionary for managing shared information objects efficiently among distributed co-workers. In order to manage these complex information objects, the proposed framework adopt product structures represented by Bills Of Materials (BOM) as stems for integrating the various information objects. Based on the proposed framework, a collaborative software development management system (CSDMS) is implemented, and the functionalities and the structure of the system are also described in this paper. The proposed system provides sufficient functionalities for the change management of information objects and the management of their relationship to other objects rather than existing system.

Keyword : Information Object, Collaborative Software Development, Collaboration, Shared Workspace

* 경상대학교 산업시스템공학부 교수
** 한남대학교 경영정보학과 교수, 교신저자

1. Introduction

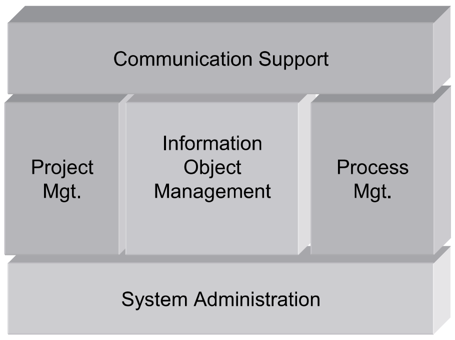
Most enterprises are struggling to change their existing business processes into agile, product-centric, and customer-centric structures to survive in this competitive global business environment. This is no exception in the software industry. Currently, since software development team members have been more geographically spread due to the globalization of business and Internet technologies, project management and communication efforts for developing high-quality software products on time are becoming more complicated.

A general software development process can be described as follows : a software enterprise organizes qualified people into a project team for developing a specific software product. During the process execution, team members collaborate with each other toward common goals. After all, a project team produces a software product, which is the goal of any project itself. In cases of distributed development, more managerial difficulties arise than with general software development because the group work is distributed across space and time. These cases require more efficient collaborative work to bring the efforts of all co-workers together in order to develop a software product. So there is a great need to establish a collaborative software development management environment, where collaboration and communication among distributed team members can be carried out effectively. This would enable software enterprises to team-up with their distributed team members to develop more innovative software products at lower cost, and with reduced time to market.

The functional requirements for a collaborative software development management system (CSDMS) can be summarized as follows [4, 13]. 1) It must provide distributed team members with a shared or virtual workspace to coordinate various activities for collaborative work, as a co-located environment provides team members with face-to-face meeting facilities. 2) It must facilitate to create, store, distribute, and share information objects such as design documents, source code, and metadata associated with software objects produced during a software development life cycle. 3) It must provide process management functions to define and implement change processes and workflows based on business rules. 4) It must provide project management functions such as project scheduling, task allocation, and status monitoring.

Among these requirements, the systematic management of information objects produced-during a software development life cycle is essential because distributed team members perform their tasks through the creation, access, update, use, and exchange of information objects such as documents and source code.

A layered functional architecture of the CSDMS focusing on information object management is depicted in [Figure 1]. It consists of three layers. The first is a communication support layer, where distributed team members can communicate with other participants for collaborative work synchronously or asynchronously. The second layer is composed of three modules, among which the information object management module is the core module, and is complemented by the project and the process management modules. The third, the system administration layer, provides common system-level administration functions such as user authorization/access control, security control, and storage management.



[Figure 1] Layered Functional Architecture of CSDMS

To develop an integrated and full-fledged system that can facilitate the manipulation of shared artifacts among distributed team members, such as that depicted in [Figure 1], it is essential to establish a framework for systematically managing the various information objects produced during the software development process.

Conventional manufacturing industries have used product data management systems to manage vast numbers of information objects for developing their complex products during the product development phase, which include product structures, drawings, technical notes or calculations, etc. In order to manage these complex information objects, current product data management systems employ product structures represented by Bills Of Materials (BOM) as stems for integrating the various information objects. As an example, [5] introduced a BOM model that can represent product configurations, assembly structures, different product views, and even engineering changes in a coherent way.

Since the BOM in the manufacturing industry is used as the fundamental scheme that integrates various product-related information into a single coherent structure, and many critical decisions are made based on how the BOM is structured [7], an integrating scheme like the BOM is also needed in software product development. Furthermore, efficient ‘software part’ management is also required to increase software development productivity by avoiding redundancies and duplications through standardization. The aim of this paper is to propose an integrated information object management framework for collaborative object-oriented software development management, and to implement a CSDMS based on the proposed framework.

The rest of this paper is organized as follows. In the next section, related work is reviewed. Section 3 presents the concept of an integrated information object management framework comprised of a so-called BOC (Bill of Class) scheme, which is a BOM in the software industry, along with the software part dictionary concept. In Section 4, the design and implementation of a CSDMS based on the proposed framework is described. Finally, the last section contains a conclusion and further research.

2. Related Work

The research on collaborative software development management is closely related to other disciplines such as project management, workflow management, CSCW (Computer Supported Cooperative Work), software process model, and software configuration management. Among research of workflow systems and software process models, Endeavors [3], an open, distributed, extensible process execution environment, was designed to improve coordination and managerial control of development teams. Its architecture explicitly supports the goal of the distribution of users and processes, and provides rich support for integration with commercial off-the-shelf tools. Endeavors supports dynamic process changes over configurable enactment models, has an event monitoring structure, and has integrated support for communication between participants. The SPADE [2] environment has much in common with Endeavors. It is based on a multi-level architecture having repository, process enactment, and user interaction levels, and provides process evolution support. These two systems have, however, insufficient functionalities for the change management of information objects and the management of their relationship to other objects.

The MILOS system [13] supports the dynamic coordination of distributed software development teams by integrating project planning and workflow technologies over the Internet. It provides functionalities such as process planning and workflow execution, and has a notification mechanism. MILOS places emphasis on the integration of project planning and workflow support. It achieves this integration by supporting Microsoft's MS Project as a planning interface for the flexible workflow engine. Moreover, MILOS has extended its scope to knowledge management systems, where development teams share their experiences and knowledge [14]. However, this system also lacks the management function of information objects produced during the software development cycle.

Research on information object management in a distributed software development environment is actively being conducted in the software configuration management area [12]. Hoek [9], and [16] have addressed the integration of conventional configuration management with software process and project management functions over the Internet. Hunt [10] and Render[17] have proposed conflict resolution methods for concurrent changes on one information object by distributed team members, but this research has focused on only change control of information objects and change process execution. With regard to software object management, the DHT (Distributed semantic Hypertext) approach [15] has modeled the relationships of information objects based on the semantic hypertext model, and has implemented a virtual repository for information objects in a distributed environment. However, this approach has only focused on the configuration management features of software classes.

Among related research in the CSCW area, Appelt [1] has proposed a shared workspace system called BSCW (Basic Support for Cooperative Work), of which the core function is document management over the Internet. This research has addressed the effects of shared workspaces in globally distributed, loosely organized groups, and information about the users' activities within their workspaces, i.e., awareness services. This research places great emphasis on the group awareness features, which provide information about what was done, when, and by whom, on document objects.

In order to develop a CSDMS satisfying the functional requirements described in Section 1, it is also significant to investigate current commercially available tools for collaborative software development. Although project management tools like Microsoft's MS Project provide the functionalities of scheduling, resource allocation, and leveling for project management, they don't support the management of information objects produced during software development, and have few communication functionalities. Most traditional software configuration management tools place great emphasis on revision control of information objects. On the other hand, they have few functionalities for project and process management. General workflow systems such as IBM's WebSphere MQ workflow [11], Fujitsu's Teamware Flow [6], and Tibco's Staffware Process Suite [18], focus on the workflow automation of pre-defined processes. However, these automation functions are not integrated with project management functions and provide no systematic information object management functions. Software enactment engines are more flexible than general workflow products in software development processes, and provide comprehensive integration facilities with other software development tools. They are, however, slightly more difficult to use compared with general workflow tools, and provide few communication functionalities. As reviewed above, currently available tools only partially satisfy the functional requirements for collaborative software development management.

3. Information Object Management Framework

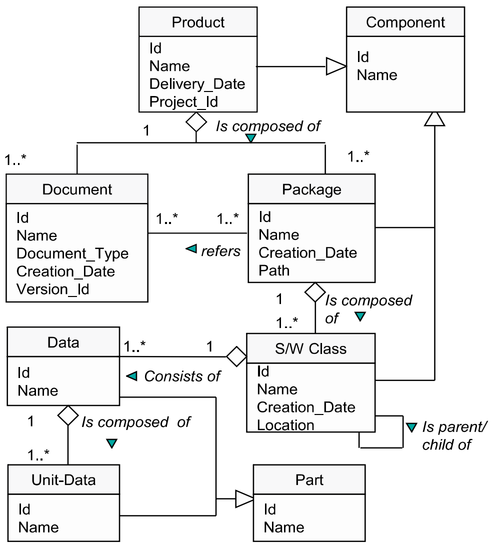
The type of information objects that are created, updated, stored, and shared during software development processes are document, source code, and metadata associated with software artifacts themselves. An information object, which comprises or describes a software artifact is related to many information objects associated with that software artifact. Because relationships among information objects are becoming more complicated as a software development process proceeds, it is therefore difficult for team members to easily understand the meaning of the relationship among them. Since this situation increases the need for more communicative effort and consequently decreases development productivity, it is essential to establish a structuring scheme for managing information objects efficiently and systematically. Highsmith III [8] has pointed out the importance of providing contextual information as well as content information for collaboration among distributed team members. Since contextual information is represented by relationships among information objects such as hierarchy, reference, dependency, and derivation, any integrating scheme must be able to manipulate these complex relationships.

During the programming phase of software development, it is common for development team members use 'software parts' such as class name and database table/field name for assembling a software product without any restrictions. As a result, the situation frequently occurs that the same name is used for defining two or more different entities, or a single entity is defined by two or more different names. Since these practices decrease development and maintenance productivity, a standard way of using software parts is needed.

A proposed framework consists of a structuring scheme and a standardized software part dictionary for managing information objects for collaborative object-oriented software development.

3.1 BOC (Bill Of Class) Scheme

A BOC (Bill Of Class) is a fundamental scheme that integrates various software product-related information into a single, coherent structure. Its structure is as follows: one object-oriented software product ('product' class) is composed of one or more packages that bundle closely related software classes and one or more documents, as depicted in [Figure 2], by the form of a UML (Unified Modeling Language) class diagram. A 'package' class refers to one or more 'document' classes that describe it. This referencing is a horizontal relationship between two classes. A document class has major attributes such as an identifier, name, type, creation date, and version number. One package class is comprised of one or more software classes, and one 'software class' class consists of multiple 'data' classes, which are composed of multiple 'unit-data' classes representing atomic data names within a program source code. For example, the 'Inventory Level' data instance, which means the inventory level of a material, is an assembly of 'Inventory' unit-data instance and 'Level' unit-data instance. A 'software class' may have a parent or a child class, and has major attributes such as an identifier, name, creation date, and location. Data and unit-data classes are further generalized into 'part' super-class, having major attributes such as an identifier and name. 'Product', 'package', and 'software class' classes are also abstracted to 'component' super-class. Consequently, a software product structure has a hierarchical relationship among its component classes. This software product structure, as described above, is called a 'BOC (Bill Of Class)' in this paper.



[Figure 2] Logical Structure of BOC (Bill Of Class)

There are four types of relationships among software product information objects, and these relationships can be systematically managed based on the BOC scheme : 1) a hierarchical relationship representing a whole-part structure, 2) a referential relationship, where one class refers to other class horizontally, 3) a dependency relationship describing a parent-child structure, 4) a derivative relationship tracing a change history of single information object instance.

Therefore, by the BOC structure, distributed team members can easily understand the relationships among information objects, and thus can use the right information objects for their development tasks. The types of relationships derived from the BOC structure are as follows :

- The software class-package relationship, which describes the where-used relation or the inverted structure of a software product.
- The super class-subclass relationship, which represents dependencies between parent and child classes.
- The document-software class relationship, which describes the hierarchical structure or inverted structure between software classes and the document in which they are described.
- The package-document relationship, which relates a package to document(s) describing that package.
- The derivative relationship, which describes the change history of a single information object represented by a version tree.

3.2 Software Part Dictionary

Logically, the structure of a software product could be represented by [Figure 2]. But the properties of the 'part' class are quite different from that of the 'component' class because the 'software class' cannot be assembled completely by using the 'data' class only at the program source code. Therefore, for the efficient management of information objects, it is necessary to divide logical BOC structures into two groups, and to manage each part by different methods. One group could be managed by a BOC scheme, which constitutes a hierarchical relationship of a software product (i.e., 'product' -'package'-'software class'). The other, which forms a 'unit-data'-'data' relationship, could be managed by a standardized software part dictionary. A software part dictionary is an information repository for the standardized data used in writing a program source code. Under the part dictionary control environment, there is an explicit constraint that the 'data' instance used at a program source code must only be made of combinations of pre-defined atomic 'unit-data.' The resulting data instance comprised of pre-defined unit-data must be also pre-defined in a software part dictionary before the actual usage at a program source code. The usage of only standardized data during programming can increase software development and maintenance productivity substantially.

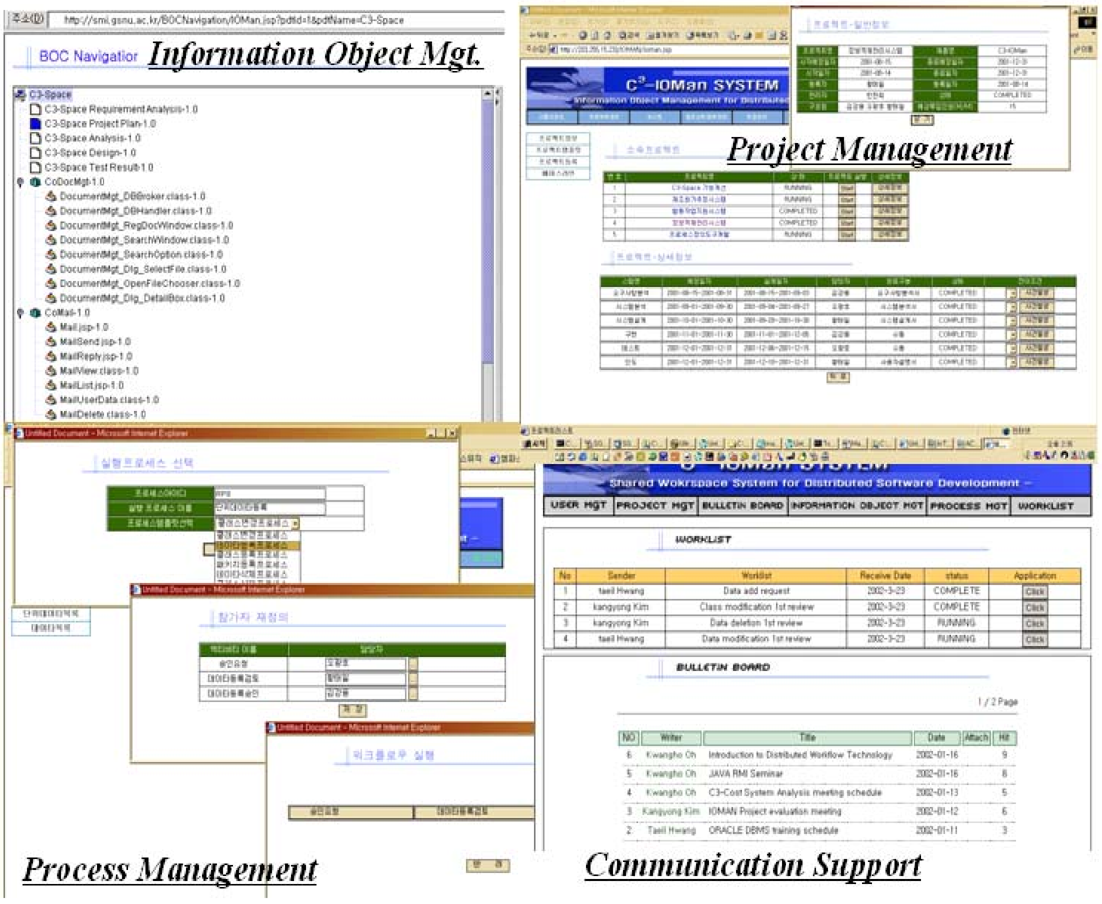
If each team member wants to use a specific data instance in his program source code, first of all he must determine whether a required data instance exists or not in the software part dictionary. If he finds required data, he can use pre-defined data without restriction. Or he must determine whether a required data instance could be assembled through some combination of pre-defined unit-data. If possible, after registering a data instance comprised of pre-defined unit-data to the part dictionary through the approval process, he can use this data instance in his program. If not, an atomic unit-data instance required for data assembly must be registered through the approval process before creating the necessary data instance. As well as registration of data and unit-data instance to the software part dictionary, a pre-defined approval process is required for all change events for information objects, such as modification and deletion during the development process. This approval process is managed by the information object change control function, which is a part of the process management of CSDMS.

4. Implementation of a Collaborative Software Development Management System : IOMAN

The basic goals of developing the IOMAN (Information Object MANagement for collaborative software management) system based on the proposed integrated information object management framework are :

- To manage the contextual information as well as content information of software artifacts
- To provide project management and monitoring tools for distributed team members
- To manage workflows for the information objects change process
- To facilitate communication and coordination among geographically spread team members

To accomplish these goals, the structure of the IOMAN system is based on the layered architecture of the CSDMS, depicted in [Figure 1]. The main screens representing modules of the IOMAN system, except for the system administration module, are illustrated by [Figure 3].



http://smi.gsnu.ac.kr/BOCNavigation/IOMan.jsp?pdtId=1&pdtName=C3-Space

BOC Navigatior

- C3-Space
  - C3-Space RequirementAnalysis-1.0
  - C3-Space Project Plan-1.0
  - C3-Space Analysis-1.0
  - C3-Space Design-1.0
  - C3-Space Test Result-1.0
  - CoDocMgt-1.0
    - DocumentMgt_DBBroker.class-1.0
    - DocumentMgt_DBHandler.class-1.0
    - DocumentMgt_RegDocWindow.class-1.0
    - DocumentMgt_SearchWindow.class-1.0
    - DocumentMgt_SearchOption.class-1.0
    - DocumentMgt_Dlg_SelectFile.class-1.0
    - DocumentMgt_OpenFileChooser.class-1.0
    - DocumentMgt_Dlg_DetailBox.class-1.0
  - CoMail-1.0
    - Mail.jsp-1.0
    - MailSend.jsp-1.0
    - MailReply.jsp-1.0
    - MailView.class-1.0
    - MailList.jsp-1.0
    - MailUserData.class-1.0
    - MailDelete.class-1.0

실행프로세스 선택

참가자 재정의

워크플로우 실행

WORKLIST

No	Sender	Worklist	Receive Date	status	Application
1	taeil Hwang	Data add request	2002-3-23	COMPLETE	Click
2	kangyong Kim	Class modification 1st review	2002-3-23	COMPLETE	Click
3	kangyong Kim	Data deletion 1st review	2002-3-23	RUNNING	Click
4	taeil Hwang	Data modification 1st review	2002-3-23	RUNNING	Click

BULLETIN BOARD

1 / 2 Page

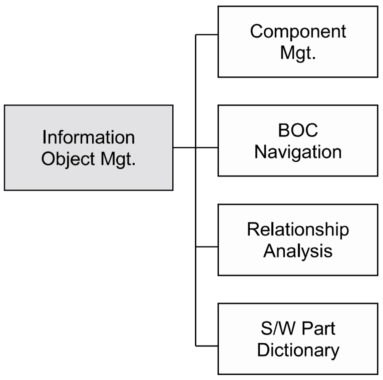
NO	Writer	Title	Date	Attach	Hit
6	Kwangho Oh	Introduction to Distributed Workflow Technology	2002-01-16		9
5	Kwangho Oh	JAVA RMI Seminar	2002-01-16		8
4	Kwangho Oh	C3-Cost System Analysis meeting schedule	2002-01-13		5
3	Kangyong Kim	IOMAN Project evaluation meeting	2002-01-12		6
2	Taeil Hwang	ORACLE DBMS training schedule	2002-01-11		3

[Figure 3] Main screens of IOMAN system

The project management module is divided into two parts. One part is for defining the process definition template required for a software development. The other part is for selecting an appropriate process definition among the predefined process templates for the execution of a specific development process, for modifying and determining the schedule and resource allocation for each process step, and for monitoring the project status. The process management module provides the capabilities of managing information object change workflows and administrative workflows. The information object change workflow is the approval process for the requested changes such as registration, modification, and deletion of various information objects. The implemented function for communication support in the IOMAN system is the bulletin board facility, where major events and information are posted to team members. By utilizing this facility, they can discuss and exchange opinions regarding any specific topic.

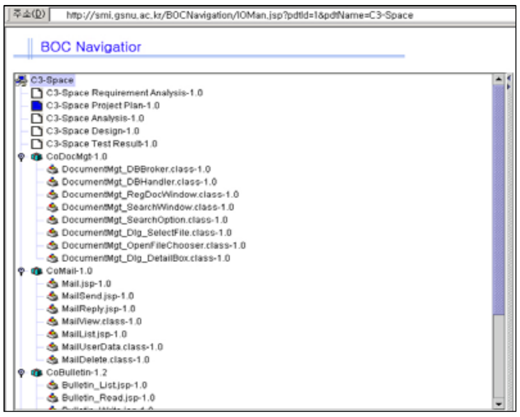
In the IOMAN system, a single shared workspace is allocated to one software project team, and only authorized project team members can access his team's workspace. Because it is essential for distributed team members to access and participate in this shared workspace more easily, this system is designed in a web and Java-based 3-tier architecture, which enhances the interoperability, platform independency, and reusability. For this architecture, the user interface is developed by using JSP (Java Server Page) and Java applets, and the application logic is implemented by using Java classes and Java beans. The operating environment of the IOMAN system is as follows : Apache 1.3.19 for a web server, Tomcat 3.2 for a servlet engine, and MS SQL 2000 for the persistent data management.

As an illustrative example, we chose the information object management module, since it is a core module where various information objects for software artifacts are managed systematically by the BOC scheme and a software part dictionary. As depicted in [Figure 4], the information object management module is divided into four major functions : 1) component management, 2) BOC navigation, 3) relationship analysis, and 4) software part dictionary.



[Figure 4] Function Chart of Information Object Management Module

The component management function manages the central repository, where contextual information is stored as well as content information of information objects within software artifacts. The BOC navigation function provides the capability of graphical viewing and navigation of a BOC structure. [Figure 5] displays a typical BOC structure of a specific software product called 'C3-Space', where different icons distinguish product, document, package, and software class from each other, and a management document is also differentiated from a technical document by icon color.

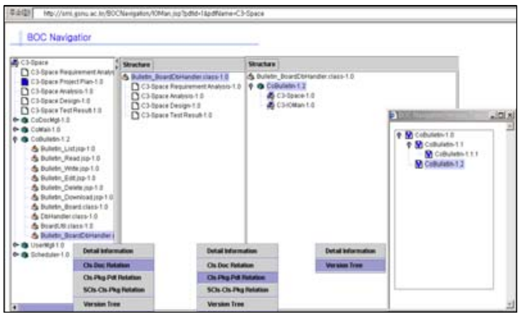


```
주소(D) http://smi.gsnu.ac.kr/BOCNavigation/IOMan.jsp?pdtId=1&pdtName=C3-Space

BOC Navigatior

C3-Space
    C3-Space Requirement Analysis-1.0
    C3-Space Project Plan-1.0
    C3-Space Analysis-1.0
    C3-Space Design-1.0
    C3-Space Test Result-1.0
    CoDocMgt-1.0
        DocumentMgt_DBBroker.class-1.0
        DocumentMgt_DBHandler.class-1.0
        DocumentMgt_RegDocWindow.class-1.0
        DocumentMgt_SearchWindow.class-1.0
        DocumentMgt_SearchOption.class-1.0
        DocumentMgt_Dlg_SelectFile.class-1.0
        DocumentMgt_OpenFileChooser.class-1.0
        DocumentMgt_Dlg_DetailBox.class-1.0
    CoMail-1.0
        Mail.jsp-1.0
        MailSend.jsp-1.0
        MailReply.jsp-1.0
        MailView.class-1.0
        MailList.jsp-1.0
        MailUserData.class-1.0
        MailDelete.class-1.0
    CoBulletin-1.2
        Bulletin_List.jsp-1.0
        Bulletin_Read.jsp-1.0
```

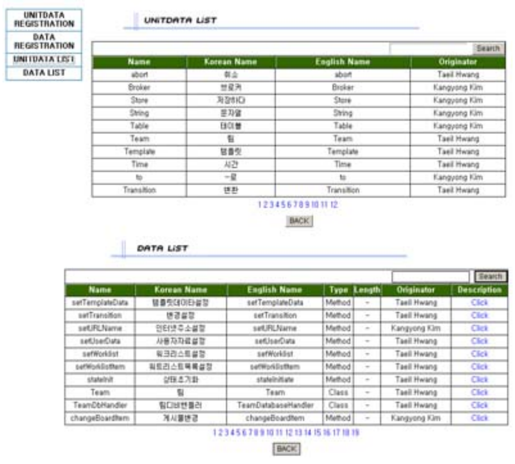
[Figure 5] BOC (Bill Of Class) Structure

The relationship analysis function displays the relationship information among the information objects graphically so that team members can understand with ease the impact of a proposed change of a software component upon the product structure. [Figure 6] displays the relationship analysis results of specific software product - the leftmost window displays the BOC of a software product (C-3 Space), at which the relationship of the shaded 'Bulletin_BoardDbHandler1.0' class is analyzed. The second window shows a class-document inverted structural relationship of a shaded item in the first window (i.e., Bulletin_BoardDbHandler 1.0). This relationship shows that the description of a specific software class, called Bulletin_BoardDb Handler1.0, can be found in documents represented as parts in a tree structure. The third window displays a class-package-product relationship of a shaded item in the second window. This relationship represents an inverted structural relationship (i.e., a where-used list) in a software product structure. This window shows that the Bulletin_BoardDbHandler1.0 class is included in a CoBulletin1-2 package, which in turn can be found at the product C3-Space1.0 and the product C3-IOMan1.0. Finally, the fourth window shows the version history of a shaded item in the third window (i.e., CoBulletin1.2 package). A version tree describes a derivative relationship (i.e., a change history) occuring within a single information object. In this example, the prior version of the CoBulletin 1.2 package is the CoBulletin 1.0, which is also re-versioned to the CoBulletin 1.1 that is further branched to the CoBulletin 1.1.1.

During the programming phase, at which distributed team members use the IOMAN system for collaborative software development, project team members have to use only the standardized data stored in the software part dictionary that manages data and unit-data. This restriction can increase the software development and maintenance productivity substantially by avoiding redundancies and duplications of data. There are many types of data managed in the software part dictionary such as class name, attribute name, method name, package name, database table name, and database table column name. [Figure 7] shows data and unit-data lists managed by the software part dictionary. As displayed in [Figure 7], data has attributes such as a name, Korean name, English name, data type, data length, originator, and description.



[Figure 6] Relationship Analysis



UNITDATA LIST

Name	Korean Name	English Name	Originator
abort	취소	abort	Taeil Hwang
Broker	브로커	Broker	Kangyong Kim
Store	저장하다	Store	Kangyong Kim
String	문자열	String	Kangyong Kim
Table	테이블	Table	Kangyong Kim
Team	팀	Team	Taeil Hwang
Template	템플릿	Template	Taeil Hwang
Time	시간	Time	Taeil Hwang
to	~로	to	Kangyong Kim
Transition	변환	Transition	Taeil Hwang

1 2 3 4 5 6 7 8 9 10 11 12

DATA LIST

Name	Korean Name	English Name	Type	Length	Originator	Description
setTemplateData	템플릿데이타설정	setTemplateData	Method	-	Taeil Hwang	Click
setTransition	변경설정	setTransition	Method	-	Taeil Hwang	Click
setURLName	인터넷주소설정	setURLName	Method	-	Kangyong Kim	Click
setUserData	사용자자료설정	setUserData	Method	-	Taeil Hwang	Click
setWorklist	워크리스트설정	setWorklist	Method	-	Taeil Hwang	Click
setWorklistItem	워크리스트목록설정	setWorklistItem	Method	-	Taeil Hwang	Click
stateInit	상태초기화	stateInitiate	Method	-	Taeil Hwang	Click
Team	팀	Team	Class	-	Taeil Hwang	Click
TeamDbHandler	팀디비핸들러	TeamDatabaseHandler	Class	-	Taeil Hwang	Click
changeBoardItem	게시물변경	changeBoardItem	Method	-	Kangyong Kim	Click

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

[Figure 7] Data/Unit-Data List of Software Part Dictionary

In the IOMAN system, since all information objects are stored at the central repository, the check-in task must be performed after an information object is created or modified. So only after the registration or change request is approved can it be checked-in as the newly released version. The check-out task is performed when information objects are requested for inquiry or modification by team members. In this case, the system checks whether this check-out is for simple use without a change or for use for a change. When the check-out for a change is requested, the system locks the requested object and prevents multiple team members from attempting to change this information object simultaneously. When a check-out for currently locked information object is requested, information such as current owner and check-out date is provided to the requester. This facilitates the task of coordination related to the locked information object between the object owner and the requester.

<Table 1> shows the functionalities of our collaborative software development management system compared to commercial systems in project management, workflow management, and configuration management fields. The proposed system covers most of functionalities required in software development life cycle as illustrated in <Table 1>.

5. Conclusions and Future Work

Recently, since cases of distributed software development and component-based development are rapidly increasing, it is essential to provide a collaborative software development management environment for reducing development time and cost. During a development process, team members perform their tasks through the creation, access, update, use, and exchange of information objects such as document and source code. Because the relationships among

<Table 1> Comparison result in functional aspect

systems / required functions	Project Management tool (MS project)	Configuration Management (Borland StarTeam)	Workflow system (IBM MQ workflow)	Proposed IOMAN system
Communication support	-	-	○	○
Information object management	-	○	-	○
Project management	○	△	△	○
Process management	○	○	○	△
System administration	○	○	○	○

information objects are very complex, and the management of software development processes requires a multidisciplinary approach, the collaborative object-oriented software development management must be based on a coherent and integrated information object management framework. This paper proposes the BOC scheme, under which various information objects are structured systematically and relationship information among them such as hierarchy, reference, dependency, and derivation is managed effectively, as well as content information of information objects. Also proposed is the software part dictionary concept, which accelerates the standardization of data used during the programming phase. Under the software part dictionary, development and maintenance productivity can be increased substantially by avoiding redundancies and duplications of data. Lastly, the functionalities and structure of a collaborative software development management system is presented through the implementation of a CSDMS.

More research is, however, still needed in the IOMAN system to add new functionalities, and to improve current implemented functions. At present, team members participating in the IOMAN system have to register all metadata and relationship information about software objects manually. This is tedious and time-consuming, so it would be better if a part of this data were extracted from source code and stored to the repository automatically. For this function, it is necessary to integrate the system with commercial development tools. In the process management module, a new graphical process modeler is also needed rather than the current text-based process definition editor. Finally, for facilitating collaborative work, communication functions among team members and awareness features must be extended and strengthened.

Acknowledgement

This research was conducted as a part of the next generation & new technology development project (Development of high-safety & light weight Al automotive parts for next generation vehicles) sponsored by Ministry of Commerce, Industry and Energy.

References

[1] Appelt, W., WWW based collaboration with the BSCW System, Springer Lecture Notes in Computer Science 1725, Berlin, Springer-Verlag, (1999), pp.66-78.

[2] Bandinelli, S. C., E. D. Nitto, and A. Fugetta, “Supporting cooperation in the SPADE-1 environment”, *IEEE Transactions on Software Engineering*, Vol.22, No.12(1998), pp. 841-865.

[3] Bolcer, G. A. and R. N. Taylor, “Endeavors: A process system integration infrastructure”, Proceedings of the 4th International Conference on the Software Process, Brington, England, 1996.

[4] Chaar, J., S. Paul, and R. Chillarege, Virtual project management for software, NSF Workshop on Workflow & Process Automation, University of Georgia, Athens, 1996.

[5] DO, N., H. Kim, H. S. Kim, Y. J. Lee, and J. H. Lee, Web-based Product Data Management and Parts Catalog Publication System for Collaborative Product Develop-

ment, IIWAS 2001, Linz, Austria, 2001.

[6] Fujitsu, Teamware Office, http://www.teamware.net/Resource.phx/teamware/index.htx, 2004.

[7] Garwood, D., *Bill of material-structured for excellence*, Georgia, Dogwood Publishing Company, 1995.

[8] Highsmith III JA, *Adaptive software development*, New York, Dorset House Publishing, (2000), pp.261-293.

[9] Hoek, A., D. Heimbigner, and A. L. Wolf, "Does configuration management research have a future?", Proceedings of the 5th International Workshop on Software Configuration Management, Seattle, WA, 1995.

[10] Hunt, J. J., F. Lamers, J. Reuter, and W. F. Tichy, "Distributed configuration management via java and the world wide web", Proceedings of the 7th International Workshop on Software Configuration Management, Boston, MA, (1997), pp.161-174.

[11] IBM, WebSphere MQ workflow version 3.5, http://www-306.ibm.com/software/integration/wmqwf, 2004.

[12] MacKay, S. A., "The state of the art in concurrent, distributed configuration management", Proceedings of the 5th International Workshop on Software Configuration Management, Seattle, WA, 1995.

[13] Maurer, F., B. Dellen, F. Bendeck, S. Goldmann, H. Holz, B. Kotting, and M. Schaaf, "Merging project planning and web-enabled dynamic workflow technologies", *IEEE Internet Computing*, Vol.4, No.3(2000), pp.65-74.

[14] Maurer, F. and H. Holz, "Process-oriented knowledge management for learning software organizations", Proceedings of 12th Knowledge Acquisition Workshop (KAW '99), Banff, Canada, 1999.

[15] Noll, J. and W. Scacchi, "Supporting software development in virtual enterprise", *Journal of Digital Information*, Vol.1, No.4 (1999), http://jodi.ecs.soton.ac.uk.

[16] Rational Software, *Simplifying the process of change-Rational Clear Case*, www.rational. com, Cupertino, CA, 2005.

[17] Render, H. and R. Campbell, "An object-oriented model of software configuration management", Proceedings of the 3rd International Workshop on Software Configuration Management, Trondheim, Norway, (1991), pp.127-139.

[18] Tibco, Staffware Process Suite, http://www.tibco.com/software/process_management /staffware_processsuite.jsp, 2004.

◈ 저 자 소 개 ◈

한 관 희 (hankh@gnu.ac.kr)

아주대학교에서 산업공학을 전공했으며 한국과학기술원 산업공학과 석사 및 자동화 및 설계공학과 박사학위를 취득하였다. 대우정보시스템㈜에서 17년간 근무하였으며 현재 경상대학교 산업시스템공학부 교수로 재직 중이다. 경남제조IT혁신인력양성사업단 부단장을 맡고 있으며 주요 관심분야는 Enterprise Modeling, 워크플로우, CSCW, 제조시스템 모델링 및 시뮬레이션 등이다.

송 희 석 (hssong@hannam.ac.kr)

고려대학교에서 경영학을 전공하였으며, KAIST에서 경영과학 전공으로 석사학위를 경영공학 전공으로 박사학위를 취득하였다. 대우정보시스템㈜에서 15년간 근무하였으며 현재 한남대학교 경영정보학과 조교수로 재직 중이다. Information System Review 편집위원을 맡고 있으며 주요 관심분야는 CRM과 데이터마이닝, 시맨틱웹, 모바일 전자상거래, 경영혁신과 정보화전략 등이다.