협동 버전제어 시스템을 위한 버전충돌 인지 및 버전병합 기능의 설계와 구현

이 병 걸†

요 약

버전관리 시스템은 한 개의 문서에 대한 여러 인스턴스들 사이의 일관성을 유지하기 위하여 버전의 탐색, 충돌 인지, 병합기능 등을 제공하는 어플리케이션이다. 하지만 기존의 버전관리 시스템에서는 협동작업 환경하에서 요구되는 개인별 히스토리 관리 및 저장, 텍스트 객체를 포함한 다양한 문서 저작 행위 객체들 간의 충돌 인지 및 병합, 그레인 크기 및 협동작업자의 역할에 기반한 문서 접근관리 기능 등은 지원하지 못한다. 본 연구에서 제시된 AID 태그 기법은 유일주소 지정기법을 사용하여 협동작업 환경하에서의 다양한 행위 객체들간의 충돌 인지 및 버전 병합을 용이하게 하고, 접근권한 테이블과 역할부여 기능을 AID 태그와 연계함으로써 작업 참여자의 역할 및 그레인 크기에 따른 문서 접근관리가 가능하도록 하였다. 또한 AID 태그 기법은 버전정보의 저장 및 관리를 위해 소요되는 파일의 크기를 줄이는 효과를 가져다 준다.

Design and Implementation of Differencing and Merging Scheme for Cooperative Version Control System

Byong-Gul Lee†

ABSTRACT

Version control is an application to maintain consistency between different instances of the same document allowing operations such as navigation, differencing, and merging. Most version control systems, however, lack the support of functionality for cooperative writing environment, such as to represent and store the history of the actions of different individuals, to effectively differentiate and merge the individual actions including the text object, and to manage different access privileges for different granularity and individuals. With the help of Activity IDentification (AID) tag and its unique addressing scheme proposed in this paper, differencing and merging become simple and effective. Access and role control is improved by associating the access right table and role assignment in AID tag. AID scheme also eliminates the requirements for large storage capacity for version information maintenance.

1. Introduction

Nowadays, it is quite common for groups of individuals to cooperate on major project, which normally involves a great deal of group writing, discussion, cooperation, and simulation, as well as individual efforts. During cooperation, many different variants of documents are asynchronously created or revised by distributed individuals. A version control system can be applied to maintain the consistency of these variants through differencing, merging or tracking of revisions. In asynchronous cooperative environment, text documents are replicated across all distributed sites. Changes or revises on text documents are first

† 정 회 원 : 서울여자대학교 컴퓨터학과 교수
논문접수 : 1999년 8월 12일, 심사완료 : 1999년 11월 23일

reflected on the local site, then on all other sites at some point in time to maintain consistency (through the differencing and merging process). However, currently available version control systems lack the support for its use in cooperative environment. That is, insufficient are mechanisms to represent and store the history of the actions of different individuals, to effectively differentiate and merge the revision history made by different individuals, and to manage different access privileges for the members of groups and different granularity, etc.

The main goal of this paper is to present techniques to manage the revision history and to enhance the traditional differencing and merging scheme to suit for cooperative writing environment. Especially, differencing and merging scheme adopts the new addressing mechanism (Activity Identification) and is refurnished in association with various access rights and granularity. With the Activity IDentification (AID) tag and its unique addressing scheme, differencing and merging become simple and effective. Access and role control is improved by associating the access right table and role assignment in AID tag. AID scheme also eliminates the requirements for large storage capacity for version information maintenance. The remainder of this paper is organized as follows. The major issues of version control system are discussed in section 2, and the review of currently available version control system follows in section 3. Section 4 proposes and describes the techniques for improving differencing and merging mechanism. In section 5, the result analysis for the proposed system is provided, and finally the conclusions are summarized in section 6.

2. Issues in Version Control

2.1 Differencing system

The fundamental component of a version control system is a differencing mechanism that finds differences among versions. Generally, there exist three different approaches for a differencing system. The first and the most popular approach is to compare plain text, which finds the longest common substring of two versions, and then the location of any changed strings is computed by a numeric distance from the common substring [1, 2]. Most of existing differencing tools, such as Diff3 [1], RCSmerge [2], and Flexible Diff [4], employs a plain text differencing method. The second approach is to use semantic differences by constructing dependency graph for each object and then merges them into one dependency graph and checks for differences if any [1]. ClearCase [12] and MKS [13] adopts this dependency graph method. However, both plain text comparison and semantic differencing methods impose huge overhead on the system as the number of versions to be merged increases. The last approach is to utilize the command history. In this approach, differencing is maintained by exchanging and comparing the histories of commands between version [1]. To support the comparison in command history, the idea of dynamic pointer is applied [3]. The dynamic pointer allows users to maintain a position set by relating the location in one version to the same location in another version. For example, if user A inserts two sentences between the second and the third sentence, and user B deletes the second sentence, then the corresponding position function for each version would be as following :

$$x : O \rightarrow A = \{ 1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 5, 4 \rightarrow 6, 5 \rightarrow 7, \cdots \}$$
$$y : O \rightarrow B = \{ 1 \rightarrow 1, 3 \rightarrow 2, 4 \rightarrow 3, 5 \rightarrow 4, 6 \rightarrow 5, \cdots \}$$

If user B had added a sentence to his version, say sentence #3, and wants to find the same location in the version which was created by user A, the position function can be applied as $p@B = 3$. By applying an inverse function such that $p@O = y^{-1}(p@B)$, the sentence position 4 in the original version is computed from the set. Then, applying $p@A = x(p@O) = 6$ gives a sentence position for user A's version. This scheme is very useful for navigating multiple versions along the version tree. However, the dynamic pointer scheme may impose some overheads on the system. For

example, if a user put a sentence in the beginning of a document, the rest of the sentence numbers would have to be updated and maintained in the same manner. In this fashion, redundant commands producing the same effect (as the history grows in length) may waste time and memory space.

2.2 Merging

Merge allows multiple versions to be joined together, producing a new version representing the union of the actions taken from the previous versions [4]. However, merging multiple versions is not trivial if conflicts exist. Conflicts arise if the objects to be merged are different (object conflict), the operations performed on the object, such as insertion or deletion of text, are not compatible (operation conflict) or the granularities of objects, such as word, sentence, or paragraph, are different (granularity conflict). In traditional version control systems, such as RCS [2] and MICROCOSM [7, 8], the conflict detection itself was not necessary since they all used a simple exclusive lock, and merging was accomplished by simply overwriting the previous version. To resolve the conflict problem, merge tool needs a mechanism for specifying merge rule to determine the priority among conflict objects. The merge rule can be applied based on multiple types of merge services : automatic and manual [1]. In automatic merging, users are not involved in merging process. The conflict is automatically resolved by the predefined rules. Manual merging requires users to complete the merging process manually. Concurrent Versions System (CVS) [6] utilizes the automatic merge if the modified object is not conflicting. If found conflict, the system triggers the manual merge. In cooperative writing environment, since users often may want to change their merging rules and services, system should provide users with flexible changes of rules and services interactively. Besides, merge rule should also provide a way to interact with different granularity. That is, it should be possible to handle various levels of granularity for merging [4].

2.3 Access control

Ellis defined the access control as "… a mechanism to determine who has the right to access certain information and in what manner" [5]. However, most traditional version control system fails to the support of this statement. That is, they do not provide the mechanism to access to different grain size of "certain information" and to discriminate different responsibility (role) of "who". The RCS system [2] and Liveware [11], for example, that is built on UNIX file system discriminate only users from accessing to the versioned file, not the object in various grain size. It simply gives users the same access rights to all objects. Traditional access control should also be modified to meet the needs for role based access control. In cooperative environments, users tend to collaborate as an assigned role such as writer, editor, reviewer or commentator. The role based access control should associate with granularity of shared object and allow individuals to specify changes of their access rights.

3. Related work

Revision Control System (RCS) [2] is the most popular public domain version control system. RCS offers a plain text differencing mechanism and automatic merging. For maintaining consistency in merging, a simple exclusive lock is used. Therefore, versions in RCS cannot be created or revised in parallel. The system neither can detect the conflict.

Concurrent Versions System (CVS) [6] is a collection of tools on top of RCS. In CVS, however, parallel work is possible since a version cannot be exclusively locked. It supports the automatic merge if modified objects are not conflicts each other. If found conflict, manual merge is triggered.

MICROCOSM [7, 8] is an open hypermedia system, originally developed for a PC environment. Like RCS, the system does not support true parallel editing on a same version. Furthermore, the lock not only freezes the selected version but also all the derived

or hyperlinked versions, thus make the system be inadequate for cooperative environment.

COOP [14] stores the complete version information in the last revision. Older versions can be reconstructed by tracking the histories backward. Merging provides a set of default rules to produce a suggested merged version and to identify conflicts. However, it does not provide a mechanism for users to change the default rules or deal with granularity change.

PREP [15] supports a mechanism for supporting roles and cognitive activities. This reduces the coordination problem by specifying responsibilities and patterns of interaction. However, PREP does not allow users to change their roles during cooperation. Each role is assigned at the beginning of the cooperation session, and users are not allowed to change their roles thereafter.

ClearCase [12] and MKS [13] are also classified as a version control system although their primary function is to provide authoring environment for application code development. They help users manage the various builds of their applications. Both systems use dependency graph for differencing and merging. However, they provide neither role-based access control nor granularity control.

Lotus Notes [9] is a well known and the most widely used general-purpose cooperative authoring system. The version control in Notes, however, is very limited. A user has no way of knowing how a version evolves because intermediate versions are always overwritten by a final version during the server's synchronization process. Since there is always exactly one official version, differencing system, conflict detection, and version management utilities are not necessary.

The common problems of the existing version control systems are as follows; Firstly, they lack mechanism for supporting true parallel editing of versions since they use a simple lock mechanism. Secondly, the text and dependency based differencing is time and space consuming. Thirdly, differencing and merging does not allow user to interactively deal with various types of conflicts and access rights in association with granularity and role.

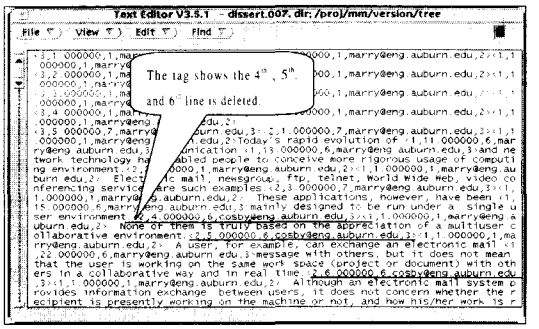
4. Design of a new Version Control system

4.1 Differencing tool

To remedy the problems described in the previous section, a new editing notation is presented by using a unique Activity IDentification (AID). AID consists of three parts; granularity level, element address, and type of operation performed. The granularity level represents the grain size such as word, sentence, or paragraph. The element address, which is used to locate and differentiate the exact position of an element, is assigned with a unique real number starting at 1.0. As a user first creates an initial document, elements are assigned with successive real numbers, e.g., 1.0, 2.0, 3.0, and so on. This initial address assignment is performed at each granularity level, e.g., word, sentence, or paragraph. After the initial assignment of address for each element, new address is determined by computing the middle point between the addresses before and after. For example, if a new sentence is inserted between the 4th and the 5th sentence, a new address, 4.5, is assigned to the inserted sentence.

The type of operation defined in AID includes 'create', 'insert', 'delete', and 'replace'. Other activities, such as 'cut' and 'copy', can be easily included as needed. The formal BNF notation of AID is represented as follows; {< (P|S|W), address, (C|I|D|R) >}* where P : Paragraph, S : Sentence, W : Word, C : Create, I : Insert, D : Delete, R : Replace. This AID tag is inserted at the front of an element every time the element is edited or updated. For example, an AID tag, <P, 1.0, C> <S, 6.7, R> <W, 3.0, R>, denotes that the third word in the sentence at 6.7 is edited with the 'replace' operation. The sentence at 6.7 that also belongs to the first paragraph is edited with a 'replace' operation, and the first paragraph is created with a 'create' operation. If the 'create' operation is found in

a tag, it means that the element has never been edited since the initial creation. (Fig. 1) shows the actual AID tag information inserted in a document.



(Fig. 1) The AID tag view

The new address scheme makes differencing be simple and straightforward since it is global, stable, and hierarchical.

1) It is global since a particular word, sentence, or paragraph from one version can be easily detected and referenced at the same address in other versions since a new address space never changes the existing address space.

2) It is stable since it always creates a new address space and never destroys or changes the existing address space. For example, a 'delete' operation does not remove the address space but simply replace the type of operation field in AID.

3) It is hierarchical since the number generation scheme conforms to the hierarchical structure such as paragraph → sentence → word. For example, a word at address 3.0 means that the word is located at the third position with respect to the sentence that it belongs.

The global and stable characteristics of the new address space saves CPU time and storage space by eliminating the needs of large database to maintain command history and dynamic pointer. AID scheme also satisfies the needs of associating the granularity into the differencing. To facilitate the conflict and role control, the AID scheme should be further refined. The details are explained in merging section.

4.2 Merge tool

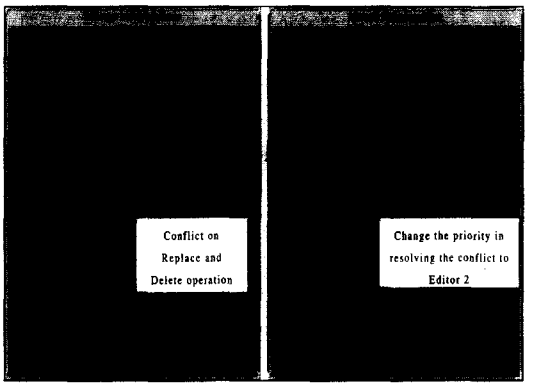
Most of the merge tools lack a flexible user interaction mechanism for specifying different levels of granularity and role. Users often want to merge their versions with different granularity or role. The merge matrix proposed by Dewan [1] solves this problem. It has a row and a column for each editing operation that can be performed on an object. The rows represent the operations of one user and the columns represent the operations of the other user(s). The entries of the matrix specify the action that has the precedence. This merge matrix is defined for each granularity level. <Table 1> shows a sample merge matrix. In the table, 'Both Operation' indicates that the new version takes the editing operation from the both versions. 'Conflict' represents that operations performed by both users are not compatible.

〈Table 1〉 Dewan's merge matrix

		User A		
	Sequence	Insert	Delete	Replace
User B	Insert	Both Operations	Both Operations	Both Operations
	Delete	Both Operations	Row Operations	Conflict
	Replace	Both Operations	Conflict	Conflict

Dewans merge matrix, however, has a fundamental problem in an n-way merging, i.e., merging n versions. A merge matrix for n users and m operations would require space complexity of $O(m^n)$, which even for two users is time consuming to fill in [1]. To remedy this problem, proposed is the use of a role-based merge table in which each dimension represents a social role rather than individual users. Since group members always assumes a certain role in normal cooperative programming activity, assigning roles in merge matrix is reasonable and natural. In addition, since role is assigned with predefined responsibility and the number of roles remains always a small

constant, n-way merge is not necessary. That is, merge can be carried out through a series of 2-way merges. Thus the space requirement for representing the merge matrix reduces to $\left[\begin{smallmatrix}K\\2\end{smallmatrix}\right] = \{k+\{k!/[2!(k-2)!]\}\} \times m$ merge matrices, where k is a number of roles. The space complexity of this approach reduces to $O(k!\,m)$. (Fig. 2) shows the actual snap shot of new merge table implemented in the proposed system. From the left figure, editor 1 and 2 are conflict on 'Replace' and 'Delete' operation. The right figure shows that the conflict is resolved by giving the priority to editor 2. Merge is primarily controlled by this merge table. Each slot in the table can be filled with a preferred role. If conflict exists and is not to be resolved automatically, the system will place the conflict information in front of the AID tag as follows; [CF//Version_name : AID_tag|Version_name : AID_tag] where CF stands for conflict.



(Fig. 2) The new merge table

For example, an editor deletes a word at position 3.25 in version *dissert.001* and another editor replaces the same word at the same position in version *dissert.002*. If two operations conflict at merge time, the following information is attached in the AID tag; [CF// dissert.001 : <W,3.25,D> | dissert.002 : <W,3.25,R>]. With this information, the user can easily find conflicts at the time of merge.

In addition, the new merging scheme provides three different roles, i.e., editor, writer, and reviewer. The role is defined by using an access right table <Table 2>.

〈Table 2〉 Access right table

Y : Yes, N : No

	Create	Insert	Delete	Replace	Comment
Editor	N	Y	Y	Y	Y
Writer	Y	Y	Y	Y	N
Reviewer	N	N	N	N	Y

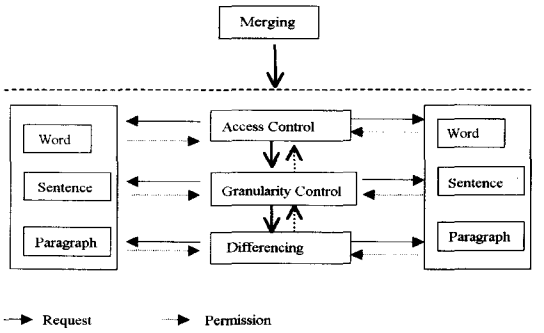
<Table 2> shows that 'Writer' has been configured with the highest priority that covers all the editing operations, i.e., insert, delete, replace, and comment. 'Editor' is not allowed to create, and 'Reviewer' can only do the comment operations. The cooperative group leader can specify or change this information.

This role specification satisfies the need of associating access rights into merge policy. To facilitate this capability, the AID tag is refined by adding role information, e.g., {<(P|S|W), address, (C|I|D|R), (**T**|**R**|**A**)>}* where T : Time of creation, R : Role, and A : Author. For example, <P, 3.0, I, 1999:03:09:13:45:33, Writer, Tom> denotes that the third paragraph is inserted by Tom as a writer, in 13:45:33, March 9th, 1999.

However, there is one problem in the role assignment. As described in previous section, users frequently change their roles during collaboration. This means that access right of each element can also be changed as the role changes in time. For example, a word that has been created by a 'Writer' can be changed by user with another role, e.g., 'Editor'. In this case, the original author may not be able to access the word unless he/she gets the editor role. This situation can be disastrous in cases when two versions are merged at a lower granularity level, e.g., word level. Once the merge is performed at the word level, the AID information of the merged word should be propagated to sentence and paragraph level to maintain the consistency. If a user who owns the merged word cannot access to the sentence or the paragraph that the merged word is contained, merge inconsis-

tency occurs. Therefore, before the merge process is performed, AID system checks the access permission, in prior, at the higher granularity levels. If the access is not allowed, merge should not be performed.

The new merge table also allows users to merge versions based on various granularity levels. Merge between two versions can be performed in word, sentence, or paragraph level, and the merge process should be aware of access rights for each element to be merged. When two versions are merged with a smaller granularity, e.g., word level, the result should be propagated to the upper level. That is, the AID tags of sentence and paragraph should also be updated. However, in this case, if the winner does not have the access permission for the upper levels, merge will not be permitted. (Fig. 3) depicts the relation of the merge process with other processes.



(Fig. 3) The relation of merge process with other sub-systems

5. Result analysis

The AID scheme is advantageous over the traditional differencing mechanism in terms of the number of operations and space. Let us assume the number of element for paragraph, sentence and word is N_p, N_s, and Nw respectively, and address generation takes only one operation (constant).

a) The number of operations for the insertion or deletion of a word :

In command history mechanism, the update for the rest of the address spaces in a document should be performed as a new word is inserted. If a word is inserted at the beginning of the document, the addresses of N_w-1 word should be updated. If the word is inserted at the second word location in the document, N_w-2 word addresses should be updated, and so on. Therefore, the average number of operations for updating other address is

$$\Sigma N_w = N_w (N_w + 1)/2 = O(N_w^2).$$

If we assume other operations such as finding the location of the word to be inserted takes only one operation, the total average operation for the insertion is bounded by $O(1+N_w^2) \approx O(N_w^2)$.

On the contrary, AID and text based differencing scheme always take a constant time for the insertion and deletion. That is, it does not need the update operation for other address spaces. Therefore, the number of operation for the insertion is always bound by constant time.

b) The number of operations for the replacement of a word :

Unlike the insertion and the deletion operation, the replace operation in command history does not require the update of other address space. The replace in AID and text based differencing scheme even does not require for the generation of a new address and is bounded by constant time.

c) The number of operations for the searching of a word :

The number of operations for searching an element is different for each scheme. In text based scheme, finding a word takes $\Sigma N_w = N_w (N_w+1)/2 = O(N_w^2)$ while AID takes only a constant time with its unique number generation mechanism. In command history mechanism, a particular word across different version can be found by following the link in the address change set [3]. If the maximum number of elements in the set is S_w and the average number of link is L, the number of operations for searching an element is bound by $O(L*S_w)$. If the total number of elements in a document is N, and $S_w \rightarrow N$, the average number of searching operation is bound by O(L*N).

d) In terms of space requirements :

Let us assume that the number of operations (insertion, deletion or replace) on a word is N and the space to store each edit script or address tag needs M bytes. In history command, with the maximum number of elements in the set, N_w, at least $\Omega(N*N_w*M)$ space is required for each word to store the performed edit history. In AID, since the insertion of a word always generate a new address tag, at least $\Omega(N_w*M)$, or $\Omega(N*M)$ spaces, where $N_w \rightarrow N$, is required for the insertion of each word. For the deletion and replace operation, a word needs at most O(M) spaces since the deletion and replace operation never generates more than one address tag keeping only one address space for each word. The same analysis applies to different granularities.

From the analysis, the AID scheme is advantageous over traditional version control mechanism in several ways :

1) The AID scheme eliminates the burden of recording and maintaining the long sequence of update information. In the command history mechanism, a newly created element (word, sentence, or paragraph) would force adjacent elements to change their addresses to maintain consistency. The AID scheme eliminates this problem by creating the address space constructively and by using this space as the basis for differencing and merging. The new scheme also saves the time waste on cross referencing in other address spaces, and makes tracking of an element along different versions much easier and straightforward (e.g., sentence #5 from one version can be found at the same address in other versions).

2) The AID scheme associates the various granularities. User can choose, view, or change any elements in versions through desired grain size.

3) AID scheme utilizes the role information and allows users to select or change the desired roles during the course of collaboration. With assigned role, the activity of each member in the group is regulated with defined responsibilities. This reduces the uncertainty of individual action, and enhances the group coordination.

6. Conclusion and future work

In this paper, a new mechanism for differencing and merging is introduced, and investigated for its advantages and disadvantages. With the help of structured AID tags, differencing and merging become easy. Its unique address scheme eliminates the requirements for large storage capacity for consistency maintenance. Access control is improved by providing the role based merge table and role assignment capabilities.

For future studies, the AID version control system should be expanded to specify more roles and granularity levels. Current design has a limit in increasing additional role information in the AID tag because adding such information requires more memory and storage space. For example, from the case study, the file size of the original text version, 13,285 bytes, is increased to 32,067 bytes in the AID version. Although the increase in tag size does not degrade overall editing performance, it may deteriorate the speed of reading and writing the tag information from and to the file. To reduce the size, the Unix file systems octal numbering mechanism for access permission can be utilized, e.g., 4 for edit, 2 for write, and 1 for reviewing permission. Any combination of file access permission can be achieved by combining these numbers. If we can specify role and granularity information in this way, defining more roles and granularities would not increase the storage space significantly. In addition, the AID address numbering mechanism can be improved by refining more intervals between two elements. Currently, any new element inserted is assigned with the middle number between the addresses before and after. Therefore, successive insertion between two elements increases the floating point by one digit for each insertion, and degrades the speed of reading and writing the tag information too. If the interval between two elements is expanded into a wider

and writing the tag information too. If the interval between two elements is expanded into a wider range, e.g., 10, the precision requirement for floating point digits can be reduced significantly. For example, if we insert a new word between the 7^{th} and 8^{th} words, the new word can be first assigned with number 7.1. The next word inserted gets 7.2, and etc. If the available slot is filled up, a new range is provided again by a factor of 10 between the two elements, e.g., 7.11, 7.12, 7.13, and etc. Defining wider range in this way can reduce the increment of the floating-point digit. Decision as to how much range is appropriate for general software development is left for future study.

References

[1] J. Munson and P. Dewan, "A Flexible Object Merging Framework," Proceedings of CSCW 94, pp.230-242, October 1994.

[2] W. Tichy, "RCS-a system for version control," Software-Practice and Experience, 17(7), pp.637-654, July 1985.

[3] A. Dix and R. Beale, "Information Requirements of Distributed Workers, Remote Cooperation," Alan Dix and Rullsel Beale (Eds.), Springer-Verlag, pp.113-143, 1996.

[4] C. Neuwirth, R. Chandhok, D. Kaufer, P. Erion, J. Morris, and D. Miller, "Flexible Diff-ing In A Collaborative Writing System," Proceedings of CSCW 92, pp.147-154, Nov. 1992.

[5] C. Ellis and S. Gibbs and G. Rein, "Design and Use of a Group Editor," Engineering for Human Computer Interaction, G. Cockton (Ed), pp.13-25, 1990.

[6] D. Coleman, Groupware : Collaborative Strategies for Corporate LANS and Intranets, Prentice Hall, 1997.

[7] A. Fountain, W. Hall, I. Heath, and H. Davis. "Microcosm : an open model for hypermedia with dynamic linking," Proceedings of the European Conference on Hypertext '90, France, Cambridge University Press, November 1990.

[8] W. Hall, I. Heath, G. Hill, H. Davis, and R. Wilkins. "Microcosm : State of the Art," Computer Science Technical Report CSTR 92-18, University of Southhampton, Southampton, England, 1992.

[9] Lotus Notes : Groupware-Communication, Collaboration, Coordination, Executive summary on Lotus Notes, Lotus Development Corp., November 1995, http://www.lotus.com/bible.

[10] S. Horwitz, J. Prins and T. Reps, "Integrating non-interfering versions of programs," ACM Transactions on Programming Languages and Systems, 11(3), pp.345-387, 1989.

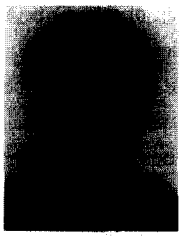
[11] I. Witten, H, Thimbly, G. Coulouris, and S. Greenberg, "Liveware : a new approach to sharing data in social networks," International Journal of Man-Machine Studies, Vol.34, pp.337-348, 1995.

[12] Http://www.pureatria.com/products/clearcase/.

[13] R. DelRossi, "Version Control hits the world wide web," Software Development, 4(12), 64-70, 1996.

[14] B. Magnusson, "Fine-Grained Version Control in COOP/Orm," Workshop on the Role of Version Control in CSCW Applications, September 1995.

[15] C. Neuwirth, D. Kaufer, R. Chandlok, and J. Morris, Issues in the Design of Computer Support for Co-authoring and Commenting, Readings in Groupware and Computer-Supported Cooperative Work, Morgan Kaufmann Publishers, pp.537-549, 1993.



이 병 걸

e-mail : byongl@cs.swu.ac.kr

1988년 University of Bridgeport 물리학과(학사)

1996년 Auburn University 전산학과 (공학 석사)

1998년 Auburn University 전산학과 (공학 박사)

1990년~1991년 시스템공학센터 연구원

1998년~현재 서울여자대학교 컴퓨터학과 조교수

관심분야 : 소프트웨어 형상관리, CSCW, 분산시스템