Automatic Link Generation and Repair Mechanism for Document Management

Takehiro Shimada
shimada@criepi.denken.or.jp

Atsushi Futakata
futakata@criepi.denken.or.jp

Communication and Information Research Laboratory
Central Research Institute of Electric Power Industry
2-11-1, Iwado-Kita, Komae, Tokyo 201, Japan

Abstract

In this paper, we describe a design and an implementation of Self-Evolving Database (SEDB), which manages various digital documents with links of five types and supports document management tasks by automating link generation and repair. The SEDB stores only links in a centralized fashion while documents are left in their native formats at their original locations. Each of the links reflects a process-oriented relationship with one of the following five types; process-flow, time-series, classification, detail, and reference. The SEDB can generate links by requirements from systems like groupware when documents have been created or modified. The links are filled in automatically by data from both the systems and the SEDB itself. The SEDB also 'repairs' dangling links which have lost one or both of their own documents by one of the following three repair operations; Link Deletion, Link Merging and Dummy Document Creation.

1 Introduction

As desk-top computer systems and networks come into wide use, people become to create and exchange many digital documents like e-mail messages or NetNews articles among several groups in daily work. One advantage of digital documents is the reusability of their contents. However it is hard to reuse them for users who are outside of the groups because they have little knowledge of the documents' contexts such as usages, purposes, creating processes, relationships with other documents and so on. They can judge neither reliability nor premise of each document and can not make appropriate use of it without its context.

Managing contexts with documents is indispensable for effective and organization-wide reusability. Contexts are essentially derived from processes in which documents have been created and/or modified. Hypermedia techniques for linking and navigating documents are suitable to represent relationships among them, but existing systems do not have much ability enough to represent complex processes running in collaborative work. Furthermore, it raises the cost of document management to make a lot of links among new documents which are created in everyday work.

WWW (World Wide Web) is the most popular distributed hypermedia system and is used to organize digital documents for information sharing over the Internet. Many types of multimedia data are accessible as dead-end nodes through WWW.

However, WWW does not have much ability enough to represent the contexts behind documents because intermediate nodes must be HTML documents and links among them are typeless. Moreover, the loss of nodes may cause a serious problem in traversing links because links are inlayed in nodes [10].

Microcosm [6] is an open hypermedia system which manages links separately from documents, and avoids two of the above problems. On Microcosm all documents remain in their native formats of word processors, spreadsheet systems and so on. Microcosm has some dynamic link features like search link, but it has only typeless static links. Therefore they do not have much ability enough to represent.

Hyperwave (formerly Hyper-G) [10] manages links separately from documents like Microcosm, and has mechanisms of automatic link generation and link consistency maintenance. Hyperwave generates glossary links and vocative links by heuristics. For the maintenance of link consistency, Hyperwave deletes dangling links when documents are deleted. In this way, surely link consistency is maintained, but users cannot reach documents beyond them.

On some other hypermedia systems like Intermedia [11], SP3 [14], gIBIS [5], rIBIS [12], links are enhanced with additional information based on models of collaborative work. They do not only have more representing ability but also entail greater cost of modeling than systems with simple links like WWW. The appropriate balance between the representing ability and the cost is important for document management.

1060-3425/98 $10.00 (C) 1998 IEEE

The Self-Evolving Database (SEDB), which has been developed at the Central Research Institute of Electric Power Industry, is designed to manage documents with links for information sharing among researchers in our laboratories. The SEDB enhances representing ability with links of five types while it keeps down the cost with functions of automatic link generation and automatic repair of dangling links. This paper describes a design, features, and an implementation of the SEDB.

2 Design of SEDB

The goal of the SEDB is to enable an effective, organization-wide information sharing of digital documents which are produced in collaborative work. The SEDB manages documents using typed links among them. It stores only links in a centralized fashion while documents are left in their native formats at their original locations. The features of the SEDB are as follows;

1. Link Typing
   In general, there are many ways to decide the numbers and meanings of link types [15]. Each link of the SEDB has one of five types such as *process-flow*, *time-series*, *detail*, *classification*, and *reference*, which are practical representations of various kinds of relationships. The type of each link is assigned to reflect the purpose of traversing the link from one document to the other. According to our observation, the purposes are classified into the following five categories corresponding to the types;
   (1) Searching or browsing documents relevant to an activity in the past.
   (2) Searching documents created by a user in the past.
   (3) Searching documents classified into the same category.
   (4) Seeing attached documents describing the details.
   (5) Other purposes.
   Each category is closely associated with the past human activity, e.g., creating a document from other documents, retrieval by a keyword, collecting documents into a directory, writing a figure or a subsection of paper, and so on. Thus the automatic link generation described below is expected to work well.

2. Automatic Link Generation
   Many systems have been developed to generate links by information retrieval methods in order to support document management [1][2][3][4][13]. Those systems only use the contents of documents and have no knowledge about the contexts in which the documents are processed and related each other.
   The SEDB provides automatic link generation facilities associated with systems in which documents are processed, e.g., workflow management systems, search engines, word-processors, and so on. Usually the system in which a document is processed has information about the document but does not have information about its relationship to others. The SEDB will provide the lacked information from the link DB of the SEDB, and generate new links.
   The networks of links generated by the SEDB are considered as logs of human activities because they have been grown automatically associated with daily work. Thus users can browse documents according to the past uses of them, and get knowledge about the past processes.

3. Automatic Repair of Dangling Links
   In the SEDB, users can remove and/or move their documents. It may cause dangling links which have lost one or both of their own documents [7]. When a document is missing, the SEDB will reorganize all links formerly connected to the missing document in order to preserve the topology of links. The SEDB has the following three basic methods and repairs dangling links using one of them; (1) Link Deletion, (2) Link Merging, and (3) Dummy Document Creation.

Figure 1 illustrates the functions of the SEDB described above. In Figure 1, each activity in the workflow-based project is represented by its corresponding *process-flow* links *A-C*, *C-E*, *B-D* and *D-E*. Plotting the graph *F* of data in the document C in the activity α is corresponding to the *detail* link *C-F*. Searching the documents *H* and *G*, which are related to the document *E*, is corresponding to the *classification* links *E-G* and *E-H*.

The automatic link generation proceeds as follows. When the document *E* has been created in the activity α, the workflow management system sends data such as the location of the document *E*, the owner of the activity α and the name of workflow to the SEDB. The SEDB receives them and then retrieves the most recently created documents in this project except the document *E* from the link DB of the SEDB. Finally the SEDB generates the new links *C-E* and *D-E* according to both data sent from the workflow management system and retrieved data from the link DB by the query.

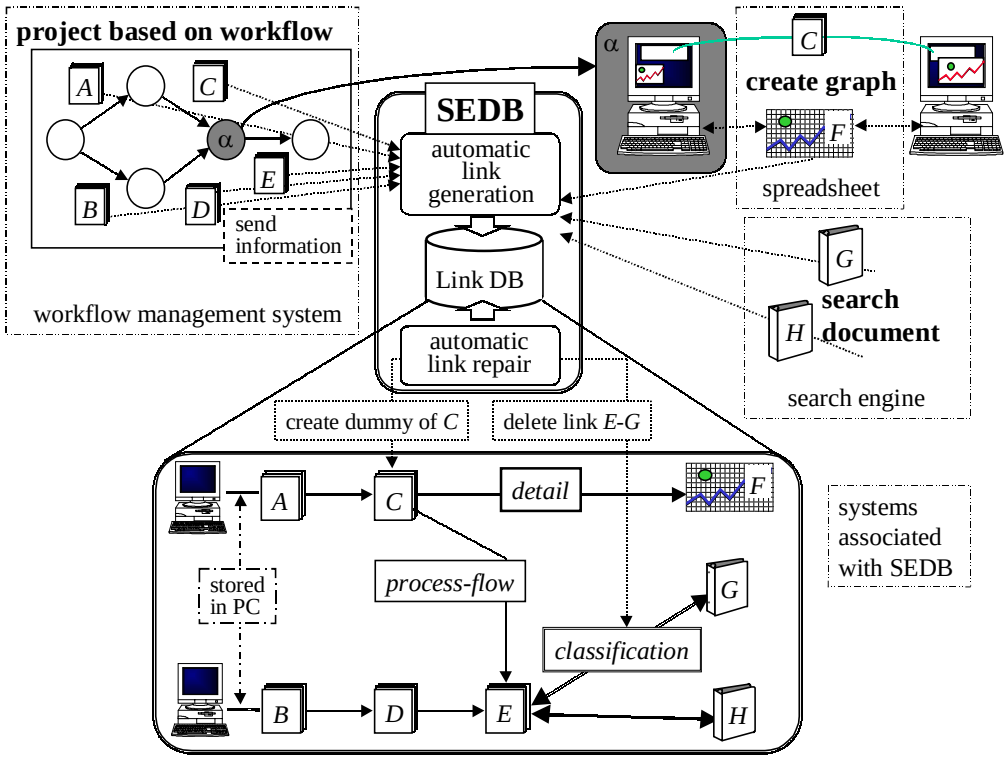

Figure 1: Image of the SEDB

If the document *G* is missing, the SEDB deletes the link *E-G* automatically because the link *E-G* has no sense any more. In the case that the missing document is *C*, to preserve the workflow structure, the SEDB creates a dummy document which contains data about the document *C* salvaged from the links *A-C*, *C-E* and *C-F*, and reconnects the three links to the dummy document.

The next three sections give a design and a mechanism of each of the three functions. Implementation issues follows the sections.

3 Link of SEDB

A link of the SEDB is an object which consists of seven fields and is stored in the link DB. Each link can connect two documents in arbitrary formats on network reachable computers. The concept of anchor, which marks an endpoint of link within a connected document, is not introduced to the SEDB. This link keeps only locations of their own documents. The seven fields of link are as follows; *source document*, *destination document*, *type*, *type attribute*, *comment*, *create time*, and *creator*. Figure 2 shows an example of the SEDB's links, which denotes that;

The slides file `C:\msoffice\powerpnt\SEDB.ppt` *stored in the PC* `brummel` *is created in an activity of workflow-based project entitled "Development of SEDB." This slides file is based on the research report* `C:\home\futakata\report.txt` *in the PC* `rocket`. *This link is created at 3/18/1997 15:30:00 by Takehiro Shimada.*

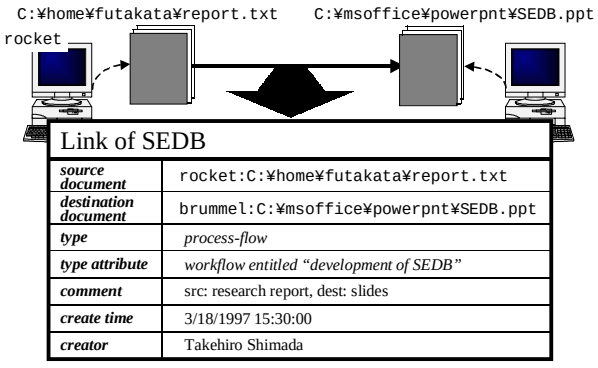

Figure 2: An example of the SEDB's links

The detailed explanation of each field is as follows;

source document / destination document

The *source* and *destination document* fields express the locations of the documents to which the link connects. The following three formats are available as the location;

File:
<*hostname*>":"<*directory*>"/"<*filename*>

HTML document:
URL

Lotus Notes [9]:
<*server*>":"<*database*>"/"<*document universal ID*>

It depends on the type of link whether the link is directed or undirected. The direction of link only indicates the role of documents with respect to the relationship denoted by the link, and users can traverse the link in both directions. The values of *source* and *destination document* are exchangeable if the link is undirected.

type and *type attribute*

The type of relationship between documents is expressed in the *type* field. The *type attribute* field has different meaning according to the type. The following five types are available for links;

***process-flow*:**
A *process-flow* link is directed and is characterized by the process in which the destination document was created from the source document. The name of the process is set as the type attribute, e.g. *workflow entitled "development of SEDB"* in Figure 2. Users may know the past activities of an organization by traversing *process-flow* links.

***time-series*:**
A *time-series* link is directed and puts the documents whose owner is the same in order of creation or modification time. The owner name of the documents is set as the *type attribute*. Users including the owner may know the past activities of the owner by traversing *time-series* links.

***classification*:**
A *classification* link presents undirected transitive relationship and is used to classify documents into the same category by connecting them. An arbitrary character string like "TO DO list", "research memo", or a list of keywords used in document retrieval are accepted as the *type attribute*.

***detail*:**
A *detail* link is directed and points to the destination document which gives explanation about any topic in the source document in detail. An arbitrary charactor string which describes the topic briefly may be set as the *type attribute*. The *detail* links are such that pointers to an explanation about technical term, a reference of paper, and so on.

***reference*:**
A *reference* link is undirected or directed, and presents a miscellaneous relationship which is not classified by the above four types. The *type attribute* includes the indication of direction, and may include a string which explains the property of the link.

comment

This field expresses the comment about the source and destination documents. This may help users to reach their desired documents. For example, Figure 2 shows that the source document is the research report and the destination is the slides file.

create time

This field expresses the time when the link was created. The SEDB adds this field automatically when the link is generated. The *create time* is used for the expiration of links.

creator

This field expresses the user who created the link. The SEDB adds this field automatically when the link is created. The owner of the process becomes *creator* when the link is generated automatically by the request from the process. The *creator* is ordinary used for access control.

4 Automatic Link Generation

This section shows the SEDB's mechanism of generating new links. The automatic link generation is performed in association with systems in which documents are processed. The communication facilities with the SEDB are required to the associated systems. The mechanism is designed to implement the facilities in the systems simply and to generate links flexibly from information from both the systems and the SEDB.

In the automatic link generation process, first, the system sends the following information (A), (B), and (C). In order to simplify the explanation, we denotes the newly generated link as *l*.

(A) Field values of Link

The system sends some field values which the system knows. The values are assigned directly to the fields of the link *l*.

(B) Link direction

This field indicates whether the link *l* becomes directed or not. Either "directed" or "undirected" is allowed as the value. This value makes sense only if the type of *l* is *reference*.

(C) Query

The SEDB retrieves the field values which are not included in (A) by this query. The query has the following form;

"("<*field11*>,…,<*field1n*>")""="{"<*condition*>","
"("<*field21*>, …, <*field2n*> ")""}"

On the *condition*, the SEDB retrieves the link from the link DB. The current implementation of SEDB allows the logical formulae constructed from the comparison of the field values of link. The max(<*field*>) and min(<*field*>) functions are also available at the most outside of the *condition*. Those functions are evaluated after other parts of the formulae have been evaluated. The values of *field2x* ($x = 1, \ldots, n$) in the retrieved link are assigned to *field1x* of the link *l* respectively.

Next, the SEDB retrieves the lacked field values from the link DB by the query described in (C). Finally, the SEDB generates new links from the field values from (A) and (B) and the rest field values retrieved on (C).

Figure 3 shows an experiment which we have made with Lotus Notes. In this experiment, there are three Notes databases A, B and C, and anyone is permitted to write documents in them. Those databases are designed to work in association with the SEDB to generate *time-series* links, each of which connects a newly created document and the last document of the same owner over the databases when the document is created. Suppose that Takehiro Shimada has created the Notes document *N* in database *B*. Then the SEDB generates a new link l_2 in the following manner.

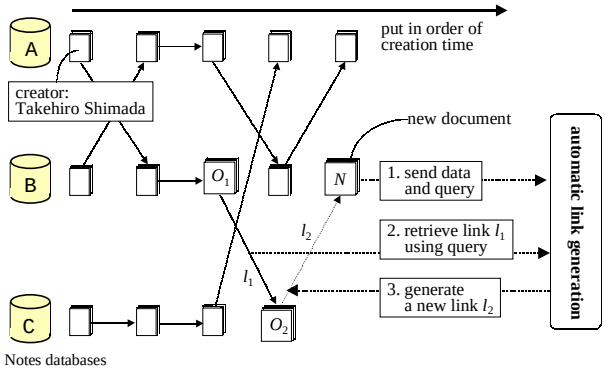

Figure 3: Experiment of Automatic Link Generation

(1) The Notes agent, which is triggered just before the document *N* is saved, sends the information shown as Table 1 to the SEDB. The query means that;

The *destination document* of the link which satisfies the *condition* becomes the *source document* of the link. The *condition* means the *type* of link is "*time-series*" and its *type attribute* is "Takehiro Shimada."

(2) The SEDB retrieves a link l_1 shown as Table 2 from the link DB using the query in Table 1. Then "document O_2 in database C", the *destination document* of l_1, will be assigned to the *source document* of the link l_2.

(3) The SEDB creates and initializes a new link, and fills the field values of Table 1 and *destination document* of l_1 in it. Table 3 shows the newly generated link l_2.

Table 1: Field values and query sent from agent

destination document	document *N* in database B
type	*time-series*
type attribute	Takehiro Shimada
direction	directed
query	('*source_document*') = {(*type* = "*time-series*" & *type_attribute* = "Takehiro Shimada") & max('*create_time*'), ('*destination_document*')}

Table 2: The Link l_1 Retrieved from the SEDB

source document	document O_1 in database B
destination document	document O_2 in database C
type	*time-series*
type attribute	Takehiro Shimada
comment	
create time	3/18/1997 15:30:00
creator	Takehiro Shimada

Table 3: The New Link l_2 Generated by the SEDB

source document	document O_2 in database C
destination document	document *N* in database B
type	*time-series*
type attribute	Takehiro Shimada
comment	
create time	3/18/1997 15:35:30
creator	Takehiro Shimada

5 Automatic Repair of Dangling Links

As users can freely modify, move, and remove their documents, it is hard to prevent the occurrence of dangling links, which have lost one or both of their own endpoint documents. Dangling links in hypermedia systems are harmful for users because they destroy the information structure of documents and relationships between them. For example, if an HTML document, which collects links to documents about a research topics, is missing on WWW, users may not reach the documents in it and the interests of the collection are lost. Thus the SEDB repairs dangling links automatically when they are found.

On WWW, when a user traverses a link which is connected to a missing document, the server or the browser only return an error message to the user. Because links are inlayed in HTML documents and HTML documents are world-widely distributed, it is hard to cope with dangling links in WWW. There are some systems like LocalWorm [8], which search for existent dangling links within a server. However these systems cannot repair them because information about the missing document is lost at the time of the document's missing. On the other hand, all SEDB's links are managed separately from documents and have information about connected documents. Therefore, the SEDB is able to 'repair' dangling links with the information in the link DB.

Four different situations where dangling links occur are proposed by Grønbæk and Trigg [7]. However we consider only one situation such that a document is unavailable by moving, removing, or network unreachability because the SEDB has no deletion operation and no anchors.

5.1 Policy of Repair

The automatic repair function of dangling link is invoked whenever a missing document is found by traversing a link formerly connected to it. At this time, all of the links formerly connected to the missing document is proved to be dangling links. The SEDB repairs only them even if other dangling links exist in the link DB because it is waste of time to find and repair unused dangling links. The automatic repair of dangling links is designed to preserve the topology of link networks because it represent the information structure in a sense. If dangling links are reconnected to different documents independently, the result networks may be different from the original networks topologically. Therefore, this function repairs all the dangling links formerly connected to the same missing document together (Figure 4).

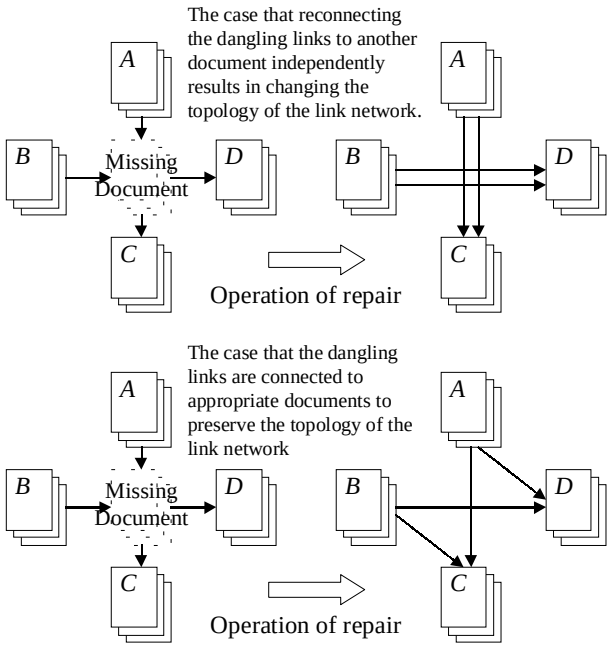

Figure 4: Repair dangling links & preserve contexts.

In the case that the dangling links are caused by network unreachability, any repair operation is not performed because network unreachability is temporary mostly.

If a document with the same name is found in the neighbor (in the upper or lower directory) of the location where the missing document was originally located, the SEDB sends a notice to the administrator and doesn't repair at the time. If the administrator confirms that the document can substitute for the missing document, the SEDB reconnects the dangling links. Otherwise, the automatic repair of dangling links is continued.

For simple cases such that each of the dangling links connected to the missing document has the same *type* and *type attribute*, the SEDB uses only the following three repair operations (Figure 5).

Link Deletion: Deleting the dangling links.

Link Merging: Merging the dangling links into several links which are not dangling. For example, the two dangling links in Figure 5 are merged into one.

Dummy Document Creation: Creating a dummy document which is substituted for the missing document and reconnect the dangling links to it.

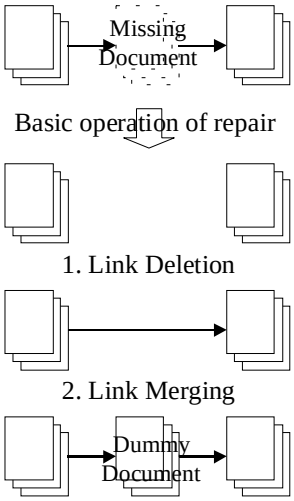

Figure 5: Example of the three repair operations

Detail explanation of these operation is described in Section 5.2.

In the above operations, the SEDB actually deletes no dangling link and only marks them as inactive links, because the missing document may be restored by the administrator. Of course, the administrator can delete them at a time.

In more complex cases such that there are more than two types of dangling links, they are classified by *type* and *type attribute*, and one of the three repair operations is applied to each classes severally. Detailed explanation is described in Section 5.2.

On the operation of Dummy Document Creation, the dangling links are reconnected to the dummy document. The advantage of Dummy Document Creation is preserving the topology of the link networks entirely. Therefore, it is adequate to apply Dummy Document Creation to the set of the links of a type like *process-flow*, because the network of *process-flow* links represents the flow of work essentially.

On the other hand, the order of documents is essential in *time-series* links, and the continuity of documents is essential in *classification* links. Dummy Document Creation preserve the order of *time-series* and the continuity of *classification*, but the increase of dummy documents causes the complexity in following links and browsing documents. Therefore, in some cases described in Section 5.2, Link Deletion and Link Merging are operated because they are more adequate than Dummy Document Creation.

Dummy documents are created not only for preserving the topology of link networks, but also for leaving tracks of the missing documents which are useful if anyone will create their substitutes. A dummy document consists of the following five items extracted from the dangling links formerly connected to it.

***missing document*:** The location where the missing document was formerly stored.
***repair time*:** The time when the repair was done.
***repair user*:** The user who traverse the dangling link and find the document is missing.
***dangling links*:** The list of the dangling links to which the repair was applied.
***comment*:** Anyone can add information about the document here.

5.2 Repair and Link Types

As described in Section 5.1, Link Deletion and Link Merging are more adequate than Dummy Document Creation for *time-series* and *classification* links. In the case that there are only *time-series* links and/or *classification* links in the set of dangling links, the procedure of the automatic repair of dangling links is the following.

1. Classify the dangling links by *type* and *type attribute*.
2. Apply an appropriate operation of the three (described below) severally for each class.

In the case that *type* of the class is *time-series* and the missing document is the source document of each of the dangling links in the class or the destination document of them, Link Deletion is operated on the class. In the case that the type of the class is *classification* and there is only one dangling link in the class, Link Deletion is operated on the class.

Otherwise, Link Merging is operated on the class. The algorithm of Link Merging on the class of dangling links $\{a_0, \ldots, a_n\}$ of *time-series* and *classification* is as follows.

Link Merging for *time-series* (Figure 6)

1. Suppose that $D_0, \ldots, D_k$ are the source documents of the links, each of whose destination document is the missing document, and that $D_{k+1}, \ldots, D_n$ are the destination documents of the links, each of whose source document is the missing document.
2. For each D_i ($i = 0, \ldots, k$) and for each D_j ($j = k+1, \ldots, n$), create a new *time-series* link whose source document is D_i and destination document is D_j.
3. Delete all the dangling links in $\{a_0, \ldots, a_n\}$.

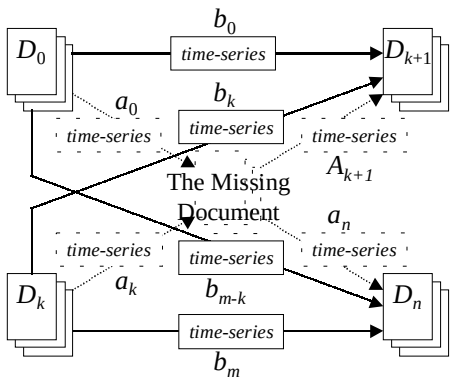

Figure 6: The operation of *time-series*

Link Merging for *classification* (Figure 7)

1. Suppose that D_0, …, D_n are the documents, each of which is the other endpoint document of the dangling link than the missing document, and that $D_{n+1} = D_0$.
2. Create a new *classification* link b_i which connects D_i and D_{i+1} for each $i = 0$, …, n. If a link whose *type* and *type attribute* is the same exists already, b_i is not created.
3. Delete all the dangling links in $\{a_0, \ldots, a_n\}$.

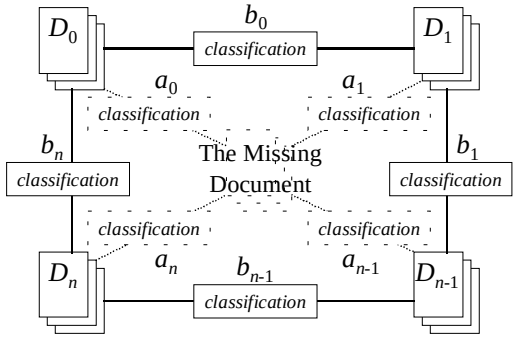

Figure 7: The operation of *classification*

For example, in the case that there are two classes of *time-series* links with *type attribute* "A", "B" and two classes of *classification* links with *type attribute* "X", "Y" in the set of the dangling links (Figure 8), Link Merging is operated on the classes of *time-series* with *type attribute* "A" and of *classification* with *type attribute* "X". Link Deletion is operated on the classes of *time-series* with *type attribute* "B" and of *classification* with type attribute "Y".

In the case that there is at least one link of *process-flow*, *detail* or *reference* type in the set of the dangling links, Dummy Document Creation is operated on the set.

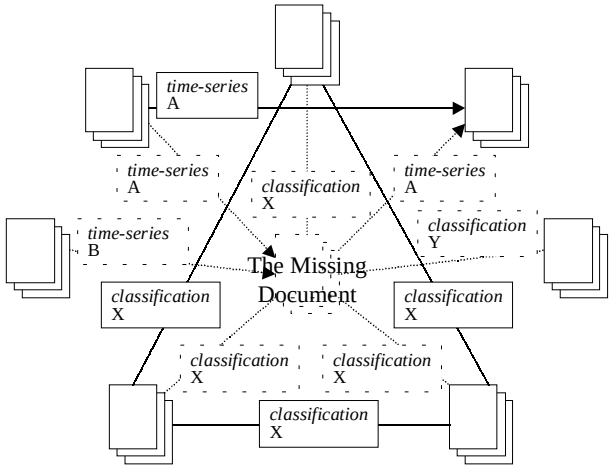

Figure 8: The operation of *time-series* and *classified*

In this case, if there are dangling links of *time-series* or *classification* type, they are reconnected to the Dummy Document (Figure 9).

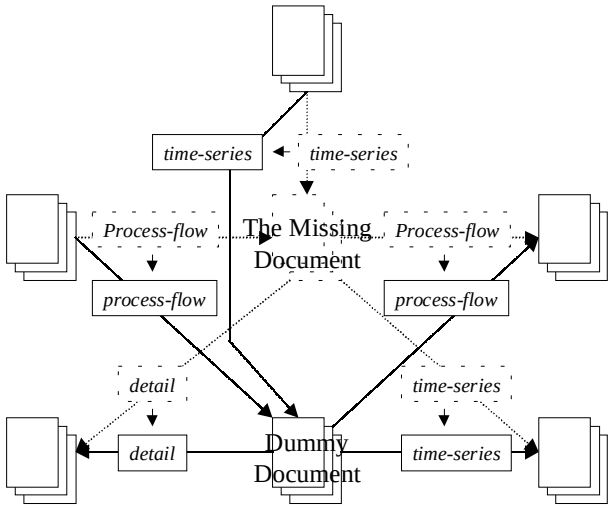

Figure 9: Repairs of dangling links with 'Dummy Document Creation'

6 Implementation

The current version of the SEDB is implemented as a database of Lotus Notes [9] on Windows 95/NT and stores links as documents of Lotus Notes.

Figure 10 is an example of the view of the SEDB. Links managed by the SEDB are listed in the window **A** and contents of a link are shown in the window **B**. This link connects the documents, one of which is identified by the universal ID `B116AB6F`… in the database

`Test\test.nsf` on the local computer, and the another is the file `F:\Futakata\ppt\Sedb.ppt` on the PC `namaneko`.

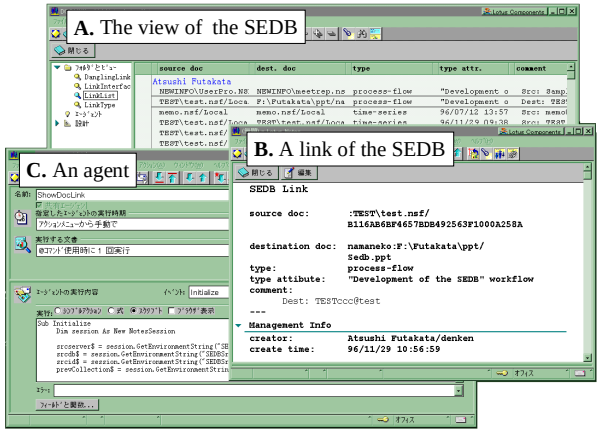

Figure 10: An Example of the view of SEDB

If the connected document is a Notes document like the *source document* of the window **B**, the value of its field whose name is either "subject" or "title" are automatically filled in the *comment* field of the link with database name.

Each function of the SEDB is implemented as an agent of Lotus Notes, which is invoked by the events such as opening a document, closing a document, or saving a document. The window **C** is a part of the agent to make Link Navigating Interface. The current version of the SEDB has the five agents described below.

Creating the Link Navigating Interface:
This agent is invoked by a user from the opened or selected document, creates the Link Navigating Interface which displays all links connected to the document, and the user can traverse a link from here. Figure 11 shows an example of Link Navigating Interface. The icon **A** in Figure 11 represents the link which connects the document to the target document, and is realized as a *doc link* or *attached file* of Lotus Notes. By single-clicking the icon, users can see the information about the link like **B**. By double-clicking, users can traverse the link and open the target document. The **C** is a button which creates and displays the new Link Navigating Interface of the target document without showing the document itself. This is useful when user wants to find out documents only by seeing the contents of link like their titles in the *comment* field.

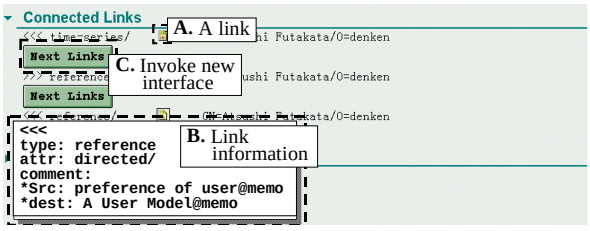

Figure 11: An example of Link Navigating Interface

Automatic Link Generation:
This agent automatically generates links using the procedure described in the section 4. The information which is sent from a system to the SEDB is transferred through the environment variables of Windows 95.

Automatic Repair of Dangling Links:
This agent repairs dangling links which is described in Section 5. When a user traverses a link from the Link Navigating Interface, and in the case that the link is dangling, this agent is invoked and repairs it. If a dummy document is created, the SEDB stores it in a Notes database for dummy documents.

Copying remote documents:
When a connected document exists on a remote PC and is going to be opened, this agent makes a copy of it on the local PC. The copy is used as a cache and is managed by a table which consists of pairs of the copied document on the local PC and the original pathname on the remote PC.

Expiration of links:
This agent automatically delete the links which have not been used for a certain time. Different deletion criterion is applied to each link type.

In addition to these, the mechanisms described below are under implementation.

Interface for Viewing Link Network:
The current version of the SEDB only has the interface which shows the relative relationships between the current document and the others. This disturbs users to recognize global relationships among documents. Thus an interface which shows the network of links is very useful for users.

Access control of document:
This mechanism offers the same access control facility of document outside of Lotus Notes as that of Lotus Notes documents.

A prototype of the SEDB is working in our local site. Currently there is only one system with the SEDB that generates *time-series* links among documents across several Lotus Notes databases automatically. Lotus Notes can only puts documents in order of creation time within one database but the SEDB enables users to

follow documents in order of creation time across databases easily.

Links of the other types are created manually now. More systems associated with the SEDB are necessary in order to use the SEDB effectively. About automatic repair of dangling links, there are only a few experience of simple test cases yet.

7 Conclusion and Future Plans

For effective information sharing, it is important to manage documents with relationships among them. This paper has described the Self Evolving Database (SEDB), which supports digital document management with links of five types, the automatic link generation, and the automatic repair of dangling links.

Although we have implemented the SEDB and have used it with only one associated system, this limited experience shows that it will be a powerful tool for effective information sharing.

More experiments with various systems are necessary for empirical evaluation of the SEDB. Thus we are now developing two systems for workflow-based project management and document retrieval, which will run in association with the SEDB.

Controlling growth of the amount generated links is useful for good performance. Therefore, we plan to enhance the expiration function prior to organation-wide use of the SEDB.

Acknowledgement

The authors thank our colleagues in Communication and Information Research Laboratory, Central Research Institute of Electric Power Industry for using the SEDB and giving us useful comments on it. We also thank the referees for their helpful comments.

References

[1] M. Bernstein. Link apprentice. In *Proceedings of ECHT '90*, pp.212–223, 1990.

[2] M. H. Chignell, B. Nordhausen, F. Valdez, and J. A. Waterworth. The HEFTI model of text to hypertext conversion. *Hypermedia*, Vol. 3, No. 3, pp. 187–205, 1991.

[3] V. Christophides and A. Rizk. Querying structured documents with hypertext links using OODBMS. In *Proceedings of ECHT '94*, pp. 186–197, 1994.

[4] P. Clitherow, D. Reicken, and M. Muller. Visar: A system for inference and navigation in hypertext. In *Proceedings of Hypertext '89*, pp. 293–304, 1989.

[5] J. Conklin and M. L. Begeeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, Vol. 6, No. 4, pp. 303–331, 1988.

[6] H. C. Davis, W. Hall, I. Heath, G. Hill, and R. Wilkins. Toward an integrated information environment with open hypermedia systems. In *Proceedings of the Fourth ACM Conference on Hypertext*, pp. 181–190, 1992.

[7] K. Grønbæk and K. H. Trigg. Design issue for a Dexter-based hypertedia system. *Communication Of ACM*, Vol. 37, No. 2, pp. 40–49, February 1994.

[8] J. Klayder. Localworm a script which detects dangling links in local disk space. `http://www.muc.edu/cwis/person/klayder/hypertext/worm/local_worm.html`, 1995.

[9] Lotus Development Cooperation. Lotus Notes R4 programmers guide, 1996.

[10] H. Maurer. *Hyperwave: The Next Generation Web Solution*. Addison-Wesley, 1996.

[11] N. Meyrowiz. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. In *OOPSLA '86*, pp. 186–201, 1986.

[12] G. L. Rein and C. A. Ellis. rIBIS: A realtime group hypertext sytem. In *Computer-Supported Cooperative Work and Groupware*, pp. 223–241. Academic Press, 1991.

[13] J. Robertson, E. Merkus, and A. Ginige. The hypermedia authoring research toolkit (HART). In *Proceedings of ECHT '94*, pp. 177–185, 1994.

[14] J. L. Schnase, J. J. Leggett, D. L. Hicks, and R. L. Szabo. Semantic data modeling of hypermedia associations. *ACM Transactions on Information Systems*, Vol. 11, No. 1, pp. 27–55, 1993.

[15] T. Trickel. What is a link type & how many are enough. `http://www.ronan.net/~ttrickel/hypertext/usentlnk.htm`.