A Hypertext System to Manage Software Life Cycle Documents

Pankaj K. Garg and Walt Scacchi

Computer Science Department,
University of Southern California,
Los Angeles, CA 90089-0782

ABSTRACT

DIF is a hypertext system which helps integrate and manage the documents produced and used throughout the life cycle of software projects. It was designed for use in the System Factory, an experimental laboratory created at USC to study the development, use, and maintenance of software systems. DIF provides an interface to a hypertext-based information storage structure and to a structured documentation process. A hypertext of software information is built by the software engineers over eight life cycle activities. Salient aspects of DIF include integration of documents within and across several projects, facilities for browsing a hypertext of software documents, support for parallel development of documents, interface to various software engineering tools, and facilitation of documents reuse.

1 Introduction

Documenting software systems is a necessity arising from the fact that not all relevant information about the system can be embodied in the code for the system. The larger the software system the more critical the problems of documents consistency, completeness, traceability, revision control, and efficient retrieval.

In [1] Balzer envisions an operating system environment based on objects and relationship between objects as opposed to the conventional file based systems. We have constructed a Documents Integration Facility, DIF, that uses this philosophy for the development, use, and maintenance of large scale software systems life cycle documents. In DIF we consider segments of software documents as the objects to be stored, processed, browsed, revised, and re-used. Links explicate the relationship between the objects. The objects are stored in files and the relationships between the objects are stored in a relational database, resulting in a persistent object base. This facilitates the reuse of documents. Software engineering tools are provided to process the information in the objects. Through judicious use of links, keywords, and structure of information, the users of DIF can alleviate problems of traceability, consistency, and completeness. Through RCS [25], DIF provides revision control. Through the Quel query language for the Ingres DBMS [18], DIF provides efficient retrieval of documents. Through the Unix mail system, DIF provides communication of structured messages between project participants. Through language-directed editors, DIF facilitates the development of software descriptions in several formal languages. Through software engineering tools for functional and architectural specifications, DIF helps the analysis of the formal descriptions.

To effectively manage the documents in a large scale software process, it is necessary to develop an understanding of what needs to be documented. To this end, we use the System Factory documentation method, as posited in section 3. The System Factory is a six year old laboratory project at USC, set up to experiment with novel ways of software project management and with the development, use, and maintenance of innovative software tools [20,9]. The uniqueness of the System Factory method of documentation is the careful blend of organizational and technical concerns that it advocates. An ongoing research effort is elaborating on the concerns to be addressed in designing such a method [14,15].

2 Software Documents in a Hypertext

DIF is a system for integrating and managing the documents produced and used throughout a software life cycle – requirements specifications, functional specifications, architectural designs (structural specifications), detailed designs, source code, testing information, users and maintainers manuals. DIF supports the information management in large software systems where the information presented in natural language text is substantial. It is an example of an Engineering Information System (EIS): software intented to support information management for the design and development of an engineered artifact. Through the development process, all relevant information concerning the target software system is stored in textual objects as nodes of a *hypertext* [6]. A hypertext is a storage structure where information is stored in the nodes of a graph. Links between nodes in a hypertext allow efficient browsing of the information. Attributes attached to nodes provide mechanisms such as information filtering. No restriction is put on the nature of information in the nodes. Hence the same hypertext may contain a node of natural language text and a node of program code. This is the main advantage that hypertext systems have over conventional database management systems, document bases, and knowledge bases. DIF has been used to integrate and manage the life cycle documents of over a dozen software systems (in the System Factory) totaling well over 100K lines of code, developed by teams of 3-7 people each. More than 40 megabytes of software life cycle descriptions are currently managed by DIF.

In a typical software life cycle, information about the process

and the product is diffused among several individuals, and it is easy for information to be lost, and for 'information bottlenecks' to be created. DIF encourages storage of relevant information about the process and the product in an easily accessible manner. The philosophy underlying DIF is two-fold. First, it is necessary for an effective EIS for software systems to prescribe the information that needs to be stored, to ensure the completeness and preciseness of the information. Secondly, for the EIS to be generally applicable, it should be possible to change the information requirements prescribed by the system according to the needs of the setting in which it is used. DIF combines these two points of views.

2.1 Comparison to other Hypertext Systems

Hypertext is a novel way of managing textual information which emerged in response to the shortcomings of traditional text management techniques [23]. DIF is in some ways similar to currently available hypertext systems. These include commercially available systems such as Guide [4], Softlib [24] and research projects such as SODOS [14], TEXTNET [27], PlaneText [6], Notecards [28] and Neptune [7]. All these share the goal of managing textual information.

An obvious difference between DIF and other hypertext systems is that DIF is geared towards facilitating information management in the software process. It has the facility to define the activities of a software process in a precise manner such that the appropriate information can be easily produced, used, and revised. DIF has been integrated with several software tools in the System Factory to provide an integrated Software Engineering Environment. For instance, the functional specifications of a system can be entered through a Gist editor which is syntactically tuned for the Gist specification language [2], and processed through the Gist Analyzer [5], through the interface provided by DIF. Through interface with RCS [26], DIF provides a revision management facility.

DIF allows the documentation of multiple projects to be stored in the same hypertext. It allows linkages between documents across and within projects. It maintains project-user information for access control. Keywords associated with the nodes in the hypertext allow the browsing of documents within and across projects.

The closest project to DIF is the SODOS project designed by Horowitz and Williamson [14]. There are key differences which set the two systems apart. SODOS managed documents for a single project only. The revision mechanism provided in SODOS was limited to the user defining the revision numbers for parts of the documents stored. SODOS was not built as part of a software engineering environment and as such did not provide interfaces to any software development tools.

TEXTNET, NoteCards, and PlaneText support the notion of building hypertext systems to support the management of textual information. Because they are not oriented towards software documents, their concern for relevancy and completeness of information, maintaining different versions at the same time, and providing access to other tools to process the information contained in the nodes, is minimal.

Neptune is designed as an EIS but the level of detail that it handles is different from that of DIF. Neptune considers information at the level of functions and procedures, much as was done in the PIE environment [12]. Hence, there are no provisions for handling multiple projects in Neptune.

3 System Factory Documentation Method

In the System Factory the software process is broken into activities, each activity culminating in the production of a document. The eight documents that emerge from the System Factory method are described below in turn:

Requirements Specification: This document describes the operational and non-operational requirements of the system being developed. Operational (testable) requirements outline the system's performance characteristics, interface constraints, quality assurance standards, and human factors. The operational requirements are defined such that they can be traced through the design and implementation of the system. Non-operational requirements outline the organizational resources available to support system development, the package of resources being built into the target system, forethoughts about the system's development life cycle, assumptions about the system operation in its target environment, and expected changes in the system operational requirements over the life of the system.

The requirements specification document is written in natural language, for which DIF provides an interface to text editing and formatting systems. DIF provides mechanisms such as keyword association and linking which help the user in browsing through the requirements. By defining a *form* for the requirements document, the manager of several projects is able to standardize the contents of the requirements document to the level of sections and paragraphs (see section 4.1). By judicious choice of keywords and links the requirements can be set up such that tracing the operational requirements through the design and implementation of the system is facilitated.

Functional Specification: This document details the computational functions the desired system is to perform in terms of the objects of computation, their attributes, attribute value ranges, object/attribute relationships, the actions that manipulate them, constraints on the actions, global stimulus-response monitors, and the system agents that organize and embed these into a system-environment. Formal functional specifications are written in the specification language Gist [2,5]. This promotes the incremental development, refinement, and rapid prototyping of operational system specifications.

A functional specifications is developed incrementally [3]. First, an informal narrative specification is given. Next, the objects (both active and passive) in the target system and its environment are described in a graphical form. Finally the formal description of the objects and agents is given in Gist. DIF recognizes the formal Gist text and allows the user to send it for processing through the Gist specifications analyzer and simulator. It also activates the Gist language-directed editor automatically.

Architectural Specification: This document describes the interconnection structure of abstract system modules with defined data resource interfaces, arranged in a way which facilitates parallel detailed design and implementation. This is described in the NuMil language [17,18]. System timing and concurrency constraints are also formally defined in this document.

DIF provides access to the NuMIL editor and processing environment which allows the definition of modules and their resource dependencies as well as the checking and tracking of changes in the modules. It also provides access to a structure visual-

izer which graphically displays the module interconnections. For more details on the NuMIL language and its processing the reader is referred to [17,18].

Detail Design Specification: This document describes (in Gist and NuMIL) the behavioral algorithms and system dependent operations consistent with the computational modules and resource interfaces configured in the architectural specification.

Source Code Document: This contains the source code of the target system which reflects the structure of the system as detailed in the above documents.

In the System Factory source code is usually written in the C programming language. DIF provides access to the C-compiler, lint, lex, yacc, dbx for efficiently transforming the detailed design developed in the previous stage to an executable code. An interface with *Make* [10] allows the user to define configurations of systems.

Testing and Quality Assurance Document: This document stores software test cases that specify how the operational requirements can be traced to test case runs of the system to validate its performance. Keywords and links can be used in DIF to link the test cases to the operational requirements.

User Manual: This document describes the commands, error messages, and example uses of the system in a standard users' manual format.

System Maintenance Guide: This document describes how the system can be enhanced, how its performance can be better tuned, known system bugs, and porting constraints.

Through this documentation method, coupled with DIF, encyclopedic volumes of software life cycle information is produced, organized, and stored for subsequent browsing, re-use, and revision.

4 DIF Usage

The organizational structure supported by DIF, in the System Factory, is shown in Figure 1. In the System Factory the project manager prescribes what needs to be described in each document. There are potentially several projects in the factory at the same time. Several software engineers work on each project.

Corresponding to the type of users in the system factory, DIF allows two modes of operations: a super user mode and a general user mode. The two modes can be compared to the database administrator and end user respectively. In the super user mode, the user defines the factory structure (what projects are in the factory and who is responsible for which), and the structure of the documents (what needs to be documented). In the general user mode the user exploits DIF to create, modify, and browse through the information hypertext. There are two levels at which the general user can operate, (1) at the information level, and (2) at the structure-of-the-information level.

4.1 Forms and Basic Templates

The super-user defines the *forms* and the *Basic Templates* (BTs) in the factory. One of the concerns of the System Factory was to ensure that all the projects have the same structure of documents. Thus each document is defined as a *form*. A form is a tree structured organization of *BTs* to be filled with information. For example, in the System Factory, the requirements specifications form is as shown in Table 1. Such forms provide a way of defining

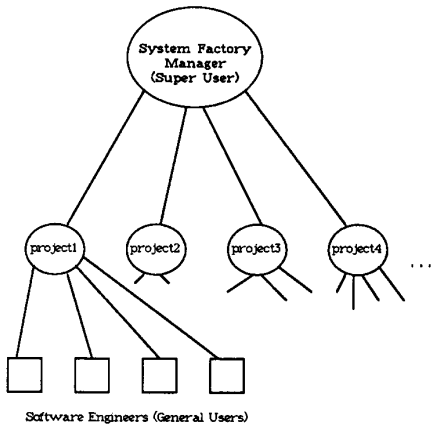

Figure 1: System Factory Structure

the software process that is to be followed by the projects in the setting.

The super user defines each form only once and all the projects inherit that form. This allows standardization of documents across projects. The intent of creating forms is that the task of the software engineers becomes one of filling in a form for documentation instead of designing their own structure for the documents. This is reflective of most of the organizational policies of software development in industrial settings.

When defining the BTs the super user also defines the nature of the information that needs be given in each BT. This entails providing the attribute of the BT as being one of:

- •Narrative •NuMIL •C Code
- •Graphical •Gist •ExeC

4.2 Project Information

The super user provides project information. Project-related information in DIF consists of a list of projects and the software engineers working on them. This information enables DIF to check the read/write privileges of the users. General users are allowed to modify only the information related to their projects. There is no restriction on reading the information of any project. The super users have read/write privileges for all information.

4.3 Information Level

Facilities are provided in DIF for the user to enter, modify, and use the information required by the forms as dictated by the super user. Language-directed emacs-like editors are provided for all the formal languages that are used in the System Factory [21]. The general user enters the information in BTs without worrying about the files that need be created. DIF automatically generates a filename depending on the project and the BT (see details of the implementation below).

Whole forms can be stored in RCS for backup purposes. There are the following functions supported by DIF, (through RCS)

1. Checking in of a form. The user can ask DIF to check-in a form into RCS.

Section Number	Section Heading
1.	Overview and Summary
2.	Problem Definition
2.1.	Technology in Use
2.2.	System Diagram
2.3.	Theory of System Operation
2.4.	Intended Applications
2.5.	User Skills
3.	Operational Requirements
3.1.	Performance Characteristics
3.2.	Standard Interfaces
3.3.	Software Quality Assurance Plans
3.4.	Software Portability
3.5.	User Orientation
4.	Non-Operational Requirements
4.1.	Resources Available for Development
4.2.	Package of Resources Built Into the System
4.3.	Forethoughts about the System's Life Cycle
4.4.	Assumptions about System Operation
4.5.	Expected Changes in Operational Requirements
BT Number	**BT Heading**

Table 1: Requirements Specifications Form

2. Checking out of a form. The user can check out a whole form from RCS.

Options such as retrieving revisions through user defined identifiers, cut-off dates, etc. are available through the interface. This is an example of 'interface transparency' that DIF provides for the tools that it interfaces to (section 5.1).

Request to process the information in a BT through a software tool can be made within DIF itself without entering the operating system. For example, if a BT contains C code, the user can request its compilation. Some such requests are handled by the Emacs editor [25], some are built into DIF (those which require the service of a System Factory tools as opposed to a Unix tool).

Interfaces to nroff/troff, spell, etc. provide the user with text processing environment akin to the documenters workbench [8].

4.4 Structure–of–Information Level

The structure–of–information[1] level allows the general user to navigate through the hypertext of information that is stored in DIF.

The user can navigate through the information in a project in the following ways,

1. **Links:** The user (super or general) can define links between BTs. The links are similar to the links allowed by most hypertext systems. Hence a browser of the hypertext can add *annotations* to the currently visited section. The operational requirements of the system can be linked up to the code/modules that support the capability. There are two key differences between the definition of links prevalent in hypertext literature [6] and that in DIF:

   - Links define semantic relationships between existing nodes. Except for annotation links, links are precluded from creating new nodes.
   - Links are allowed to be *operational links*. This readily supports the cases where executable descriptions need to be linked to the source code. For example, a C code BT can be linked to the object code BT that represents that code. Such a link is defined in DIF as an operational link. Visiting that link results in the execution of the linked BT. Arbitrary shell procedure attachments to links will be supported in future versions of DIF.

2. **Keywords:** For each BT the user can define keywords which describe the semantics of the information contained in that BT. The decision to allow for user defined keywords rather than providing automatically generated keywords was based on the results of studies reported by researchers in information science [20].

   DIF stores the keywords associated with BTs in a relation (see implementation description below). This allows the user of DIF to use the querying facilities of Ingres for navigating through documents using keywords. For example, the user can look for all BTs (within and across projects) which have a particular keyword, list the keywords of a BT, search for BTs which have keywords satisfying a pattern, etc. Standard functionalities are provided by DIF for the user not trained in Quel [19].

   An interesting feature of DIF is that it allows the reader of documents also to create keywords of their own. This allows new personnel in a project team to quickly tune the documents to their needs.

3. **Forms and Configurations:** A Form is a tree–structured organization of BTs. This provides the user with a convenient way of viewing the documents relating to each software process activity.

[1] This is different from a 'database schema'. In a schema the structure does not change with the information, whereas here the structure is dependent on the currently defined links.

To fully utilize the potential of the hypertext of information in DIF, the user can define his/her own *configuration* of BTs. A configuration is similar to a form, except that it is not enforced on all projects but is associated with the individual user who is browsing the documents.

Configurations can be defined, not unlike forms, by defining the constituent BTs. Configurations can also be defined on the basis of the trail which a user has followed while browsing through the information hypertext. Configurations are mainly used as a mechanism to print hardcopy documents, much like the *path* facility suggested by Trigg [27].

The user information space is restricted to the project currently being 'visited' by the user. To use the information of another project the user has to explicitly visit that project. This means that the user cannot use the information level commands on the information of projects other than the one that is being visited. Structure–level information is available regardless of which project is being visited. This is done to avoid the risk of the user getting lost in the information space [6]. As an additional guidance, the current project and BT are displayed in the main menu (see section 5.1).

4.5 Integration of Documents and Parallel Development

DIF provides several features which allow users to view information related to a software system in an integrated manner within and across projects. The documents are organized in a tree of Unix directories and files as shown in Figure 5. Even though the user of DIF enters information in DIF at the level of files, all the routine file management (e.g., the creation of directories for related documents) functions are handled by DIF.

This integration is invisible to the user of DIF. For instance, when entering information for the "operational requirements" of the system, the user does not have to create the file for storing the text associated with the operational requirements; this chore is automatically handled by DIF. The user need only be concerned with creating or manipulating software descriptions without being concerned about how they are stored or where. This provides an object oriented environment of persistent descriptions rather than simply a loose collection of files and directories.

DIF allows the engineers in the System Factory do develop parts of documents in parallel without worrying about the integration issues. Hence person A could be writing the operational requirements of the target system while person B is writing the non-operational requirements. The individual efforts are automatically merged in the same hypertext.

5 Implementation of DIF

At the heart of the implementation are the Unix file system and the Ingres database management system. The file system provides a repository for the textual and graphical information whereas relations in Ingres are used to store structure–of–information and project level information. In this section we first outline the support environment that DIF provides. We then discuss the structure of the file and database systems.

5.1 Organization of DIF

The basic idea in DIF is to provide an environment such that all the life cycle activities can be done through DIF itself. In this sense DIf can be considered a Software Engineering Environment [13]. With the progress of the target software system through the various life cycle activities, DIF allows a uniform interface to access the appropriate tools as necessary, e.g. a functional specification analyzer, NuMIL Processor. In the interfaces to these tools, it supports the notion of 'interface transparency' i.e. DIF provides unobtrusive use of the tool that it interfaces to, while providing mechanisms that the tool itself lacks. In addition, DIF provides hypertext-based mechanisms to support the management of information.

Figure 2 shows the organization of DIF with respect to the other tools in the System Factory.

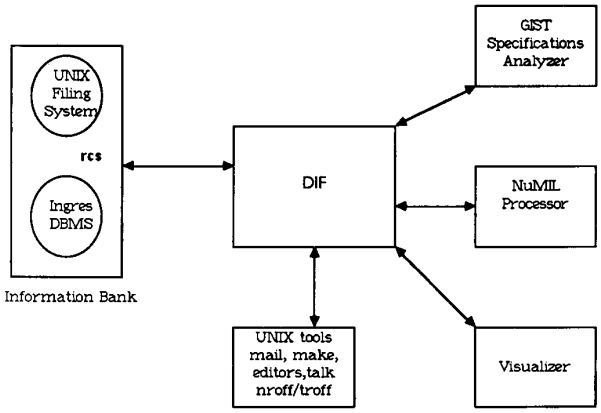

Figure 2: Organization of DIF

The basic set of tools comprises [21]:

1. A Gist Specification Analyzer and Simulator facilitates the development and use of the formal functional specifications of software (sub)systems under development;
2. A Module Interconnection and Interface Definition processor that the supports the design and evolution of multi-version system (module) configurations described in the NuMIL language;
3. An EMACS-like language-directed editing environment for constructing and revising structured documents and system description languages such as Gist, NuMIL and C; and
4. A Systems Visualizer which graphically presents system configurations;
5. Unix tools such as Rcs, Make, Spell, Nroff/Troff, Talk and Mail. The interface to the mailing system helps individuals to coordinate their activities by structured messages consisting of BTs.

An overview of the system is best provided by detailing the capabilities of the system as seen by a user. Figures 3 and 4 show the five important menus[2] provided by DIF. The general user menu differs from that of the super user only in that the general

[2]The menus have been formatted for presentation purposes

user does not have access to the functionalities of menu 2. Menu 1 is the main system menu. There are five functionalities that DIF provides through menu 1. The *utilities* provide interface to the various tools as discussed above. The *List Contents* functionality allows the user to browse through the information level. The *Table of Contents* provides the user with part of the structure–of–information level information. More structure–of–information level support comes from the sub–menus *Browsing Utilities*, and *Printing Utilities* and through the interface to *Ingres*. To guide the user about his/her current location in the hypertext, the menu displays items such as the current project being visited, the BT which is currently active, number of BTs that the user has updated in this session (forming a *trail* for printing hardcopies), and whether the output of text is being automatically processed through a formatter or not.

Menu 2 provides the functionalities that only a super user can execute. These include defining new forms for the System Factory, adding, deleting or renaming a project, adding or deleting members from projects.

Menu 3 shows the mechanisms that DIF provides for hardcopy rendition of documents. Interesting aspects include printing: the BTs visited in a session, all the Bts of a form, selected BTs of a form, and the documentation of an entire project. These are some of the mechanisms for defining *configurations* of BTs (see section 4.4).

Menu 4 shows the utilities that DIF provides to support annotation linkages. Annotations are provided in hypertexts to allow

Working project : PPSS		****DIF - Version 1.0 ****
Current BT:1.1.4	*# Updated BTs:5*	*Formatted output:y*
	Utilities	
Revision Management	NuMil processor	Edit BT
Form editor	Gist processor	Mail BT
Format form	Query Ingres	
	List Contents (pg)	
Project	Form	Basic Templates
	Table of Contents	
Project Names	Form Names	BT Headings
	Misc.	
Change to new project	UNIX shell	Change current BT
Formatted y/n		
	Submenues	
Printing utilities	Linkage utilities	Browsing utilities
Super User Functions		

Figure 3: Menu 1 – Main menu

Menu 2: Super User

Define a form
Delete a form
Modify a form
Add a project
Delete a project
Rename a project
Add members to a project
Delete project members

Menu 3: Hardcopy Rendition

Print project
Print form
Print BT
Print n selected BTs of a form
Print selected visited BTs
Print the headings of the BTs of a form
Print keywords associated with a form
HELP
Exit
Change the current BT

Menu 4: Annotation Links

Create annotation link
Delete annotation link
Modify annotation link
HELP
Change the current BT

Menu 5: Browser Menu

List attribute links
Browse via attribute links
List keywords of current BT
List keywords of current form
List keywords of another form
List BTs having a keyword, within project
List BTs having a keyword, across projects
List BTs in current form
List BTs in other form
Initiate sequential trace
Skip BT (advance to next BT)
List annotation BTs for current BT
Display current BT
Display annotation BT
Change the current BT
HELP

Figure 4: Menus 2,3,4, and 5

the facility for 'scribbling' on an on–line text. This menu provides facilities for creating, deleting, or modifying annotations, while the user can browse through the annotations using the regular browsing mechanisms (menu 5). Creating an annotation link causes a new BT to be created which stores the text in the annotation. This is the only link in DIF which allows the creation of a new node. Any user is allowed to create annotation links and BTs for any BT.

Menu 5 is the main menu which gives the user the ability to browse through the structure–of–information level. The user can browse the hypertext using keywords, forms, or links. Facilities are also provided for following the in–order traversal through a forms' BTs.

5.2 File Management in DIF

DIF builds a file structure (on Unix) which models the System Factory structure (Figure 1). Figure 5 shows an example file structure that is built. At the root is the System Factory directory which contains directories for each project. Under every project directory there is a directory for each form defined in the factory. Under the directory for each form there are files for the BTs in the form. For example, Figure 5 shows part of the requirements specifications directory in the System Factory. There are files which represent BTs 1(Overview and Summary), 2.2(System Diagram), 3(Operational Requirements), and 4.5(Expected changes in the Operational Requirements). All the projects in the System Factory have the same forms. The BTs of the requirement specifications form are listed in Table 1.

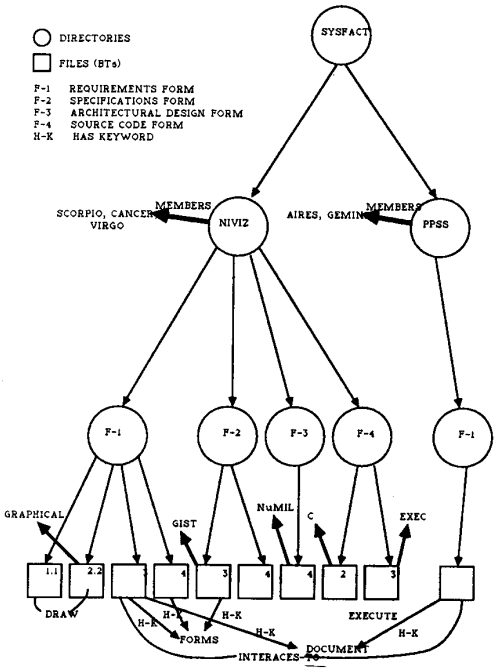

Figure 5: Hypertext of Software Documents

5.3 Database Interface

DIF builds a database using the Ingres DBMS, to store the structural and semantical information of the documents. In the following we describe the relations defined for this purpose, giving examples from Figure 5.

1. *Forms.* Forms is the mechanism by which DIF enforces standard documentation across different projects in the System Factory. The information is stored in a relation as:

Form Name	BT No.	BT Heading
Requirements Specification	1	Overview and Summary
Requirements Specification	2.2	System Diagram
Requirements Specification	3	Operational Requirements
Requirements Specification	4.5	Exp. Chgs. in Oprn'l. Requirements
Functional Specification	3	Active Objects
Functional Specification	4	Passive Objects
Architectural Specification	4	NuMIL Code
Source Code Document	2	System Code
Source Code Document	3	Executable System

2. *Keywords.* A set of keywords can be associated to each BT. The keywords are stored in the database using the following relation:

Project Name	Form Name	BT No.	Keyword
NiViZ	Requirements Specification	3	Forms
NiViZ	Requirements Specification	3	Document
NiViZ	Functional Specification	3	Forms
PPSS	Requirements Specification	3	Input
PPSS	Requirements Specification	3	Document
NiViZ	Functional Specification	4	Forms

This allows the users of DIF to create semantic links between related documents within and across projects.

3. *Text Type.* A type can be associated with each BT to define the kind of text contained in that BT. Currently the following text types are defined:

   (a) Narrative — Natural Language Text.
   (b) GIST — Functional specifications in the Gist specification language.
   (c) C — Program code in the language C.
   (d) NuMIL — Design Specifications in the NuMIL language.
   (e) Graphical — Graphics.
   (f) ExeC — Executable program.

   This information is stored as the relation:

Form Name	BT No.	Type
Functional Specification	3	GIST
Architectural Specification	4	NuMIL
Requirements Specfication	2.2	Graphical
Source Code Document	2	C-code
Source Code Document	3	ExeC

   A BT is allowed to have only one type associated with it. This is another example of the standardization that can be enforced through DIF, as the super user can initially define the types of the relevant BTs.

4. *Links.* Links are a mechanism by which different BTs can be semantically linked more strongly than through keywords. A link is defined by the source and the target BT, and an 'type' of the link. Currently defined link types include:

   (a) COPY : A copy link type indicates that the target BT is the same as the source BT. This is similar to the 'link' provided in the Unix file system. This is an operational link type which makes sure that the semantics of the link are not violated.
   (b) EXECUTE : The source BT is one of Gist, NuMIL, or C, and the target is either empty or an Exec. This is also an operational link, which when 'traversed' processes the target BT appropriately. Hence, if the source is C code, visiting the EXECUTE link for it will result in the execution of the object code linked to it. It takes care of the currentness of the target with respect to the source, regenerating the target if necessary.
   (c) DRAW : The source can be any type of BT but the target is always a Graphical type. Visiting this type of link draws the figure that is associated with the source text.
   (d) ATTRIBUTE : The link is a semantic link between the two BTs. The name of the link binds the source and target BTs with that relationship. To be meaningful, the user should define the name judiciously, much as when the user is defining keywords.
   (e) ANNOTATION : Signifies the link between a BT and an annotation made on the contents of the BT by some

reader.
This information is stored in the relation:

(Source) Project	(Source) Form	(Source) BT no.	(Target) Project	(Target) Form	(Target) BT no.	Type/Attribute
NiViz	Src. Code Doc.	2	NiViZ	Src. Code Doc.	3	Execute
NiViZ	Reqs. Specs.	1	NiViZ	Reqs. Specs.	2.2	Draw
NiViZ	Reqs. Specs.	4.4	NiViZ	User Manual	1.1	Copy
NiViZ	Reqs. Specs.	3	PPSS	Reqs. Specs.	3	Interfaces-to

5. *Security.* DIF maintains a relation in Ingres which identifies the members of a project.

Project Name	Member
NiViZ	Scorpio
NiViZ	Cancer
NiViZ	Virgo
PPSS	Aires
PPSS	Gemini

Users defined as members of a project have full read/write privileges to the information of that project. Users who are not members of a project have only read access to the information of that project. Super users have full privileges on all information.

6 Discussion

Tools Interface. DIF provides interfaces to various software engineering tools and it is easy to add new tools to the system. With the development of the Unix operating system the usage and development of tools has had a strong push. The Unix philosophy has been to provide loosely coupled tools and use them either independently or coupled together through piping mechanisms. This approach has been fruitful but with the plethora of tools that are commonly available now, efficient use of the tools cannot be made unless there is a structured way of accessing the tools. Therefore, systems like DIF, which organize the tools according to the usage of the tools will improve the usage of the underlying tools and hence the productivity of software engineers. In designing such interfaces it is important to support *interface transparency* such that the interface does not hide any of the original functionalities of the tools.

Consistency, Completeness and Traceability. An obvious concern for software documents management has been to provide mechanisms which will help maintain documents consistency and completeness over time, and provide mechanisms for tracing the operational requirements through the documents [14]. In SODOS consistency was managed by triggering an event when one of two related documents were modified, and completeness by checking if any part of the documents was empty. DIF provides a strong form of consistency maintenance through the 'Copy' link, which maintains exact copies of two BTs. In addition, DIF provides mechanisms to facilitate the maintenance of consistency and completeness, instead of trying to judge consistency and completeness by itself. Therefore, if users wish to find out which BTs are related to a changed BT, they can do so simply through the hypertext browsing facility, but DIF will not try to list all the changed BTs each time a change is made to a BT. Similarly completeness can be checked by printing out the forms and seeing which BTs are empty. The same philosophy is used for the traceability mechanisms. DIF provides mechanisms such as links and keywords which allow the users to build a traceable collection documents, but does not do any automatic generation of links or keywords.

Documents Re-use. Documentation of systems is a time consuming process and it is imperative that we find ways of re-using documents. DIF is a first step in this direction as it provides a hypertext based persistent repository of document objects with facilities for efficient browsing and retrieval. Hence related BTs can be located and re-used. In an empirical study done in the System Factory [3], DIF was found to be frequently used for retrieving and converting old documents into new ones, specially when the old documents were from a closely related project.

Scaling up. A major concern while developing DIF was to develop a documents management system which can support large scale software systems engineering. To this end, we have tried to make judicious choices between what information goes into the database and what information goes into the file system. Most of the textual information (except keywords) is stored in the Unix file system. In fact we have exploited the structural mechanisms of the Unix file system to encode structural information of the documents. The information that is stored in the database is therefore minimal. Throughout the System Factory experiment we have found that the information (bytes) in the database system is at least two orders of magnitude less than the information in the file system. This requires further experimentation but if the results are at all indicative, DIF has been able to achieve the kind of storage distribution that we wanted (and which can be supported by existing file and database management systems).

On-Line Information. The thrust of the DIF philosophy is to shift the medium of information exchange in the software process from paper to on-line. This has the obvious advantages of being a dynamic medium in the sense that documents can be modified very easily. On the other hand, paper documents have advantages such as physical feel, and ability to scribble. Also, one can read them sitting in a reclining position, etc. DIF has taken a double edged approach to this by providing (1) sophisticated mechanisms for hardcopy rendition of documents, and (2) hypertext mechanisms for on-line text management . We can potentially get the best of both worlds by not having to worry about storing paper documents and at the same time being able to use paper documents when desired.

User Interface and Multi-media Documents. Some desirable functionalities of a documents maintenance system are not provided by DIF. We have yet to experiment with multi-media document management, as is considered by projects such as IRIS [29]. The current implementation does not provide a fancy user interface as has been the goal of most researchers in hypertext systems [6]. Our driving concern has been the need to focus on the underlying information base and the supporting functions necessary for facilitating the effective production and usage of documents in a typical large scale software development effort.

Multi-Project Support. As mentioned in the beginning of the paper (section 2.1) DIF is unique amongst the currently defined

hypertext systems for software documents in that it provides support for multi–projects through the same hypertext. This is an advantage in large software project environments where related subsystems are developed almost independently and the efforts have to be later merged together. Through the use of appropriate attribute links and keywords, the structure–of–information level information in DIF can be used to tie together otherwise independently developing projects.

Revisions The revision management facility in DIF is provided through an interface with RCS. The interface provided allows users to do revision management on complete forms. At a time only one form is active, and the database reflects the structure–of–information relating to that form. DIF, however, does not provide mechanisms to do revision management of the information in the database. This results in a situation where the information in the file system has revision trees [26] but the database system has no corresponding structure. Revision management combining the database and the file system are being planned for future versions of DIF.

7 Summary and Future Work

DIF is an EIS tool for software engineering which supports the production and usage of documents in the system life cycle. Experiences with DIF over the past two years in the System Factory have convinced us of its utility.

A key motivation for DIF is to automate parts of the routine chores which reduce the productivity of software engineers. To this end, DIF provides a uniform interface where the software engineers deal with the objects to be manipulated, viz. sections/subsections in documents rather than directories and files in an operating system. Balzer [1] has suggested a similar philosophy for the next generation of operating systems.

DIF, as an active medium, allows us to capture much of the information related to the design, development, use, and maintenance of a software system. The super user can change the notion of the "software process" built into DIF according to the needs of the setting in which the system is being developed. Several project documents can be easily managed in the same system with possibilities of exchanging information across projects.

We are now investigating the issues of incorporating into the system more knowledge about the activities that the participants in a software project perform [11]. In DIF the granularity at which activities are considered is very coarse such as requirements specifications, functional specifications, etc. If, on the other hand we can determine smaller grained actions then we can support them better. For instance, activities of inputting information in a BT can be considered as *Speech Acts* [22]. This means that the person, by entering the information, actually wanted to achieve a goal apart from creating a persistent object. If the support system 'understands' such actions, then it can support them better. For example, after creating a new functionality (or requirement), the user of the target system normally wants to communicate the definition to the the developers. If this knowledge is precoded in the support environment, the communication act does not have to be carried out explicitly by the users, but can be carried out by the support environment as soon as the requirement is defined by the user.

8 Acknowledgements

We would like to thank Abdulaziz Jazzar for his contributions to DIF. The contribution of all those people who helped develop and use DIF in the System Factory class of '85-86, and '86-87 is acknowledged. We thank Salah Bendifallah for comments on earlier versions of the paper.

The work reported here has been supported through research grants or contracts with AT&T Information Systems, TRW Systems Engineering Design and Development, Hughes Radar Systems Group under contract number KSR576195-SN8, IBM through the Socrates project at USC, and MDA 903-81-C-0331 from DARPA to USC/ISI. In addition, the first author acknowledges the support provided by the USC graduate school through the All-University-Pre-Doctoral Merit Fellowship.

References

[1] R. Balzer. Living in the Next Generation Operating System. In *Proceedings of the 10th World Computer Conference*, IFIP Congress, Dublin, Ireland, September 1986.

[2] R. Balzer, N. Goldman, and D. Wile. Operational specification as the basis for rapid prototyping. *ACM SIGSOFT Software Engineering Notes*, 7(5):3–16, 1982.

[3] S. Bendifallah. *Understanding Software Specifications Work: An Empirical Analysis.* PhD thesis, Computer Science Department, Univeristy of Southern California, December 1987. Forthcoming.

[4] P. J. Brown. Interactive Documentation. *Software-Practice and Experience*, 16(3):291–299, March 1986.

[5] A. Castillo, S. Corcoran, and W. Scacchi. A Unix-Based Gist Specifications Processor: the System Factory Experience. In *Proceedings of the 2nd Data Engineering Conference*, pages 522–529, 1986.

[6] Jeff Conklin. Hypertext: An Introduction and Survey. *Computer*, 20(9):17–41, September 1987. Also available as MCC Technical Report no. STP-356-86, Rev. 1.

[7] N. Delisle and M. Schwartz. Neptune: a Hypertext System for CAD Applications. In *Proceedings of ACM SIGMOD '86*, pages 132–142, Wahington, D.C., May 1986.

[8] *Documenters Work Bench.* UNIX Documentation.

[9] L. B. Eliot and W. Scacchi. Towards a Knowledge-Based System Factory: Issues and Implementations. *IEEE Expert*, winter():51–58, 1986.

[10] S. Feldman. Make - A Program to Maintain Programs. *Software Practice and Experience*, 9(3):255–265, March 1979.

[11] P. Garg and Walt Scacchi. On Designing Intelligent Hypertext Systems for Information Management in Software Engineering. 1987. To be presented at *Hypertext '87.*

[12] I. P. Goldstein and D. G. Bobrow. A layered approach to software design. In D. R. Barstow, H. E. Shrobe, and E. Sandelwall, editors, *Interactive Programming Environments*, pages 387–413, McGraw-Hill Book Company, 1984.

[13] P. Henderson, editor. *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments.* ACM, SIGPLAN notices, vol. 22, no. 1, Palo Alto, California, Dec. 9-11, 1986.

[14] E. Horowitz and R. Williamson. SODOS:A Software Documentation Support Environment – Its Definition. *IEEE Transactions on Software Engineering*, SE-12(8), August 1986.

[15] A. Jazzar. *A Model for Managing Software Documents.* PhD thesis, Univeristy of Southern California, December 1987. Computer Science Department, Forthcoming.

[16] A. Jazzar and W. Scacchi. Understanding Software Documentation Work: Product, Process, and Production Setting. June 1987. System Factory working paper 87-02.

[17] K. Naryanaswamy and W. Scacchi. Database Foundation to support Software Systems Evolution. *The Journal of Systems and Software*, 7(1):37–49, March 1987.

[18] K. Naryanaswamy and W. Scacchi. Maintaining Configurations of Evolving Software Sy tems. *IEEE Transactions on Software Engineering*, SE-13(3):324-334, March 1987.

[19] *Ingres Reference Manual.* Unix 4.2BSD Documentation.

[20] G. Salton. Another look at Automatic Text-Retreival Systems. *Communications of the ACM*, 29(7):648–656, July 1986.

[21] W. Scacchi. A Software Engineering Environment for the System Factory Project. In *Proc. 19th Hawaii Intern. Conf. System Sciences*, 1986.

[22] John. R. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, London, 1969.

[23] D. Shasha. Netbook: A Data Model for text exploration. 1985. VLDB.

[24] I. Sommerville, R. Wellend, I. Bennett, and R. Thomson. SOFTLIB-A Documentation Management System. *Software-Practice and Experience*, 16(2), February 1986.

[25] R. Stallman. EMACS: The Extensible, Customizale, Self-Documenting Display Editor. In D. R. Barstow, H. E. Shrobe, and E. Sandelwall, editors, *Interactive Programming Environments*, pages 300–325, McGraw-Hill Book Company. 1984.

[26] W. Tichy. Design, Implementation, and Evaluation of a Revision Control System. In *6th International Conference on Software Engineering*, pages 58–67, Tokyo, Japan, 1982.

[27] R. H. Trigg. *A Network-Based Approach to Text Handling for the Online Scientific Community.* PhD thesis, Maryland Artifitial Intelligence Group, University of Maryland, November 1983.

[28] R. H. Trigg, L. A. Suchman, and F. G. Halasz. Supporting Collaboration in Notecards. In *Proceedings of the Computer Supported Cooperative Work Conference*, pages 153-162, 1986.

[29] N. Yankelovich. Intermedia:A System for Linking Multimedia Documents. Institute for Research in Information and Scholarship (IRIS), Brown University, Box 1946. Providence, RI 02912.