A Hypertext-Based Tool for Large Scale Software Reuse

Burkhard Freitag

Universität Passau
Fakultät für Mathematik und Informatik
Innstr. 33, D-94032 Passau, Germany
freitag@fmi.uni-passau.de

Abstract. A hypertext-based interactive tool supporting the management of large software libraries is presented. We claim that a simple, pragmatic approach to software reuse is best suited to aid the software engineer in solving the practical problems of software configuration from reusable components. When developing the system described in this paper emphasis has been put on the semi-automatic classification and interactive retrieval of components and their descriptions. The system has been installed at the BMW automobile manufacturing facilities in Munich. First experiences show good usability and acceptance.

Keywords: software reuse, hypertext, software configuration, reusable components, software re-engineering, taxonomic classification.

1 Introduction

Component reuse is a theme of growing importance as regards the productivity as well as the quality of software development. Reuse is of course not limited to program code but covers all levels of the software production process, such as program specifications both formal and informal, documentation, source code, object code, individual notes, to name just a few.

The configuration of complex systems from reusable software components requires the search for possible implementations in increasingly large component libraries. The success of this approach strongly depends on efficient methods for finding the appropriate reusable components, which is a very hard task in practice [9]. Thus tools are needed to support this activity.

We present a simple, practice oriented tool for software synthesis that supports the configuration of complex systems from a library of reusable software components. Our prototype system SEL[1] has been developed in a joint project with the German automobile manufacturer BMW and is currently installed and running at the BMW facilities in Munich.

[1] SEL - Software Engineering Library

The basic assumption underlying the system is that software modules can be described precisely enough by a set of *attributes* such as name, problem class, atomic types, i.e. a formatted description that can be scanned for matching keywords. In addition, a problem-specific *taxonomy*[2] of tasks can be defined, according to which modules can be classified. Even if an exact description is not known the user can roughly characterize the modules of interest by specifying some classes of the taxonomy in order to restrict the search space. The system also supports the user in defining *relationships* between any two modules. Using relationships, the search for a module can optionally be extended to "neighbouring" modules. This feature allows us also to relax the naming conventions that would otherwise have to be obeyed in order to guide the search for a particular "group" of modules.

Some novel approaches to the semi-automatic determination of relationships between component descriptions are incorporated into the system that increase both the precision and the efficiency of component retrieval. The determination of hidden relationships may sometimes be the only feasible approach to software re-engineering since formal descriptions are often missing in existing libraries. Practical experience shows that our approach is powerful enough for most configuration tasks.

The rest of the paper is organized as follows: In Sect. 2 we give a motivation and list some requirements that a system supporting software reuse and re-engineering should satisfy. Sect. 3 contains a detailed description of the SEL system. In Sect. 4 some facts concerning the implementation of the protoype system are collected. In Sect. 5 a comparison of the SEL to other systems supporting reuse is given. In Sect. 6 we summarize our experiences and briefly discuss future research directions.

2 Requirements for Software Reuse Support Systems

The size of component libraries, in particular in object-oriented environments, is continuously growing. While the amount of potentially reusable components increases the problem of finding appropriate components gets more and more complex. It is no longer possible to rely on simple naming conventions or a naive scan through a library database in order to retrieve a (set of) component(s) that satisfy certain constraints. Because the software industry and software departments in other industries are faced with a huge amount of existing components that have been designed in an "ad-hoc" fashion software re-engineering has become an essential problem, too. Thus it is not enough to have systems that support the development of new reusable software components. An automatic tool must also address the generation of documentation and interface descriptions

[2] Taxonomic classification should not be confused with classification in the sense of object-oriented programming languages. The latter can also be used to guide the search for modules, but the former normally does not induce subtype relations or code inheritance.

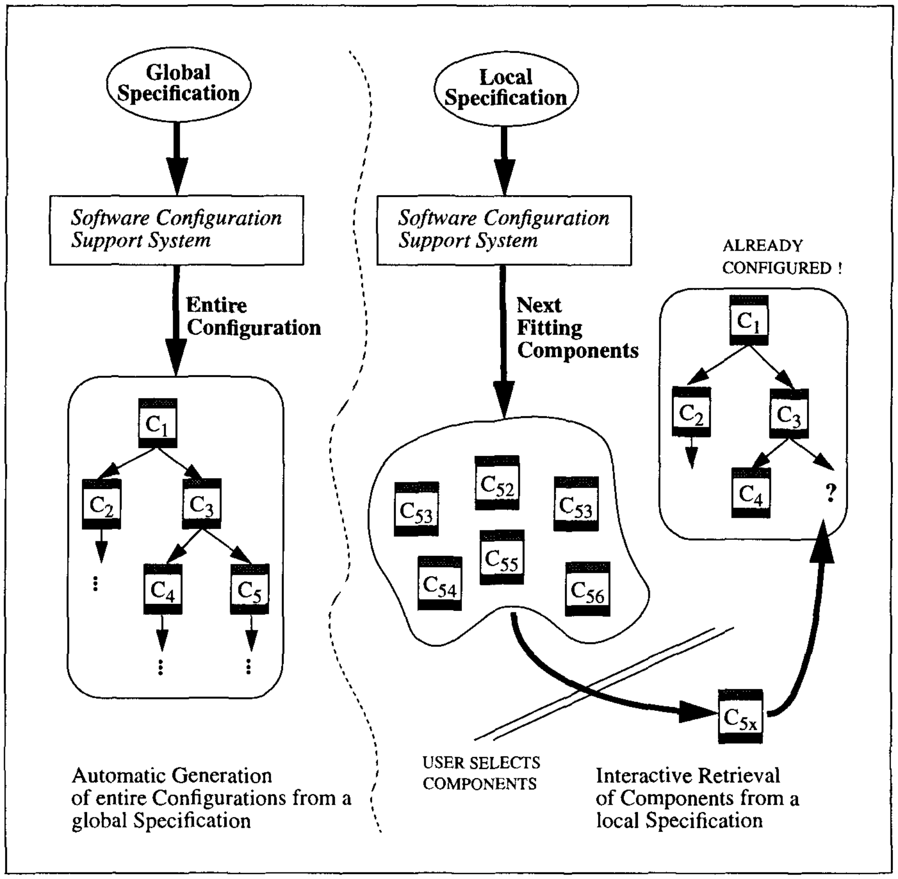
Global Specification
Software Configuration Support System
Entire Configuration
C1
C2
C3
C4
C5
Automatic Generation of entire Configurations from a global Specification
Local Specification
Software Configuration Support System
Next Fitting Components
C52
C53
C53
C55
C54
C56
C5x
USER SELECTS COMPONENTS
ALREADY CONFIGURED !
?
Interactive Retrieval of Components from a local Specification

Fig. 1. Different objectives of configuration support systems

for existing modules. These modules often differ drastically in their structure. Consequently, the adaptability of the component descriptors that are handled by the system is strongly required.

Generally, a system supporting the construction of software systems from reusable components can have two different objectives (see Fig. 1):

- Automatic generation of entire configurations, i.e. sequences of components that "fit together", from a global specification, or
- Interactive search for the set of components that fit to an already established partial configuration. In this case only local specifications governing the selection of a single component are considered.

The SEL system described in this paper belongs to the second class.

The key idea of reuse support systems [1, 2, 4, 5, 10] is the separation of component descriptions from the components themselves. A reuse support system has to address at least the following issues:

- Creation of component descriptions.
  Description templates should be adaptable to the specific class of components considered.
- Definition of expressive relationships between components.
  The classification of components should not rely on a fixed scheme.
- Support for the discovery of hidden relationships.
  For re-engineering purposes this may be the most important requirement.
- Query flexibility.
  General ad-hoc queries should be supported in addition to pre-fabricated special queries.
- Query precision.
  It does not help if the system proposes hundreds of candidate components as the answer to a query.
- Query efficiency.
  Since the class of systems we consider here typically aids the user in the local search for a set of components the query answering process should not take more than about 0.5 sec.
- Concurrent multiuser access.
- Component migration and access privileges.
  During the development of a single software component the software engineer should have exclusive access rights. After the component is released, however, it should be visible by other project members or the entire company.

Theoretically, a formal specification completely characterizes the appropriate set of modules for the given task. A weaker criterion might be type correctness or, more general, interface correctness. In practice, however, it is necessary to have selection conditions beyond type or interface correctness to guide the search for modules that are suitable for the given task [3]. From the user's point of view it may sometimes be most important to find the set of modules that offer the intended functionality [9] according to an *informal* specification.

3 The SEL component library system

3.1 Overview

The SEL system presented in following sections addresses the problems described in Sect. 2. Fig. 2 shows an overview of the various tasks performed by the system.

Ideally, a SEL user specifies his or her needs and lets the system search for a component that exactly or nearly satisfies the specification. If none such component can be found the user will certainly define a new software module that should be brought into the SEL system for later reuse. It is the responsibility of the designer of a new component to add an expressive component description to

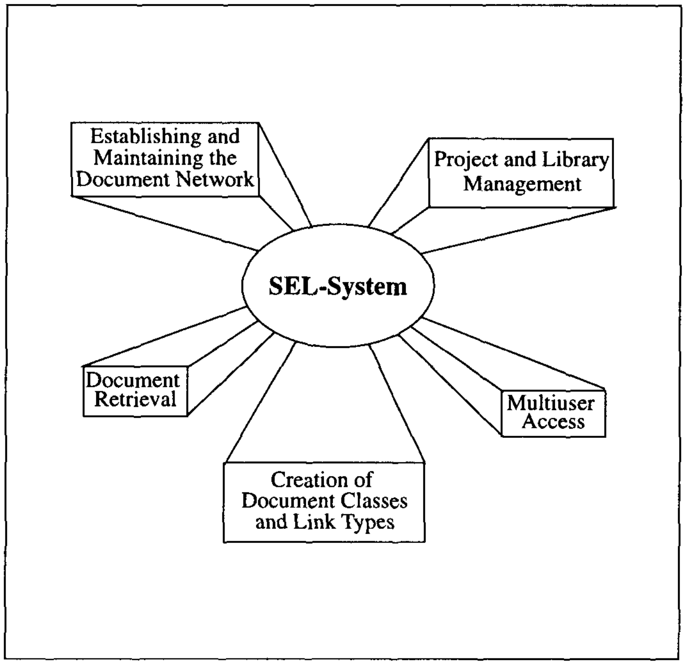
Establishing and Maintaining the Document Network
Project and Library Management
SEL-System
Document Retrieval
Multiuser Access
Creation of Document Classes and Link Types

Fig. 2. Overview of the SEL system

the SEL system. To this end, the SEL system offers tools with a graphical interface that facilitate the definition of new component classes with the appropriate descriptive framework, access privileges for the new component descriptions, and links to existing component (descriptions). Initially, new component classes and (descriptions of) new components reside in the private library of the designer. Later, when a component has been tested and approved its description may migrate to the next higher library where it can be read by other users.

3.2 The SEL data model

Documents and Document Classes Component descriptions are represented as *documents* which in turn are records with a set of attribute names and attribute values. Recall that several different descriptions may be attached to a software component, e.g. specification, informal description of the component's semantics, documentation, manual entry. However, descriptions of the same sort that are attached to components that are "similar" normally have a uniform representation. Consequently, the SEL system allows to group documents having the same structure into *document classes* which are organized in an inheritance hierarchy[3]. Document subclasses inherit the structure of the superclass (i.e. all

[3] Only single inheritance is allowed.

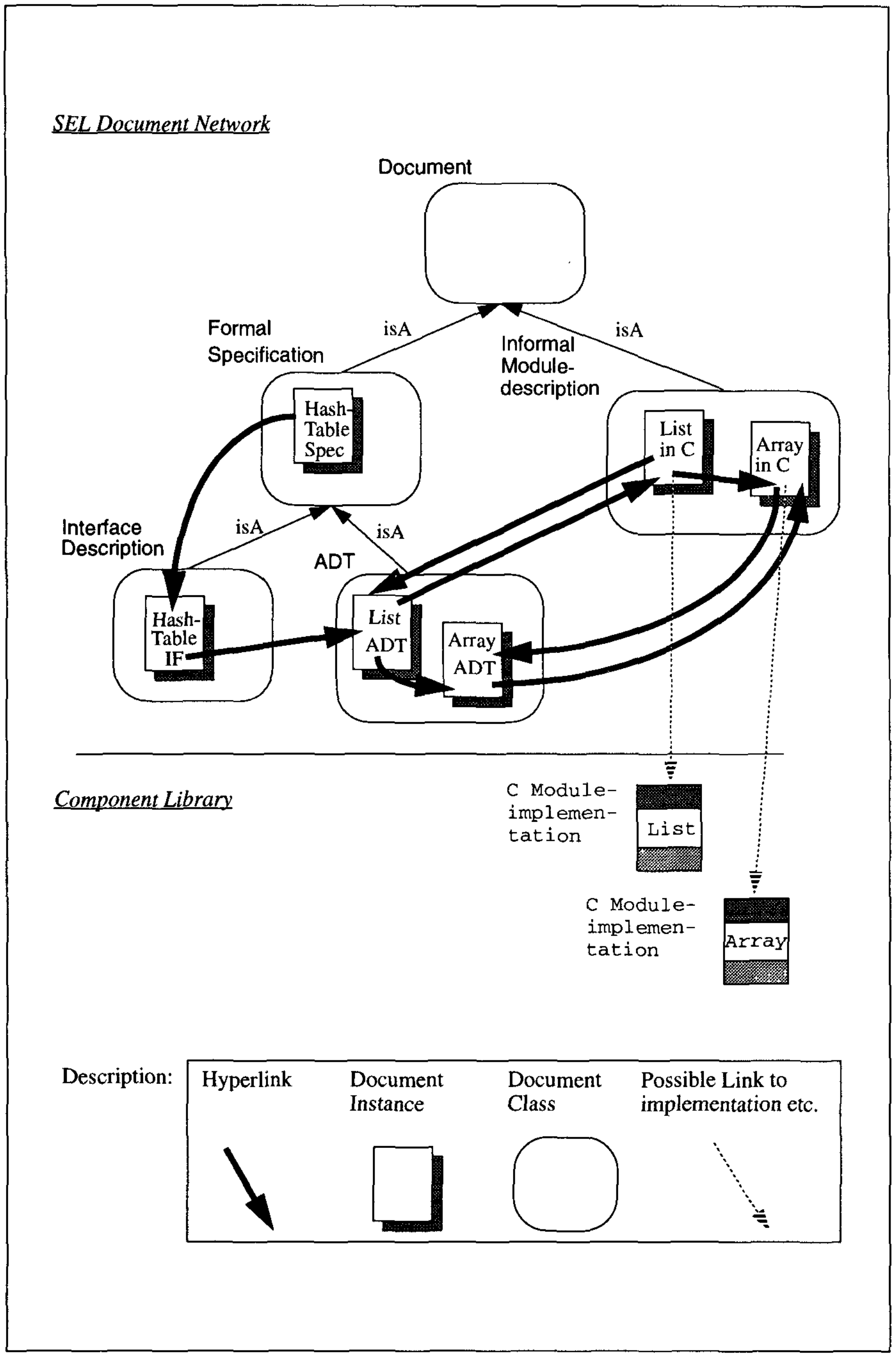

Fig. 3. A hierarchy of document descriptions

document attributes) so that documents of the subclass have a more detailed structure than those of the superclass. Note that not the software components themselves are organized in an inheritance hierarchy but rather the document classes that define the form of component descriptions. For the rest of the paper it is essential to understand that the hierarchy of document classes is to be distinguished from classes in the sense of object-oriented programming languages. Fig. 3 shows sample document classes and documents, i.e. instances of document classes. As for document classes, a document has a user-supplied name that serves as an object identifier. A document can be an instance of only one document class.

It should be mentioned that document classes can also serve as a simple means for a taxonomic classification of component descriptions according to the semantics of the components they describe. It is, for instance, possible to group all documents related to C-programs into a document class C_DOCUMENTS and similarly for LISP_DOCUMENTS.

To give an example[4] of document creation let us assume that the user intends to create a new document class MY_SIMPLE_DOCUMENTS. Let the class SIMPLE_DOCUMENTS be among the already defined document classes and let the structure of SIMPLE_DOCUMENTS be closest to the needs of the new document class. In our example SIMPLE_DOCUMENTS has the following structure[5]:

```
CLASSIFICATION:
    Name:
    Informal Description:
    Programming Language:
    SW-Engineer:
```

The user defines MY_SIMPLE_DOCUMENTS as a subclass of the document class SIMPLE_DOCUMENTS with the following additional attributes that represent relationships to other documents:

```
RELATIONS:
    uses:
    is used by:
```

The class MY_SIMPLE_DOCUMENTS is now available, and document instances may be created. Of course, attribute values have to be defined for each instance.

Relationships between documents As the example above indicates, relationships between documents can be represented by attributes that have object identifiers, i.e. document names, as their values. Documents can be retrieved by specifying some attribute values. Consequently, in the example all documents

[4] For the sake of clarity we choose a very simple example. Real-life examples, for instance those investigated in cooperation with BMW, are beyond the space limitations of this paper.

[5] Note, that SIMPLE_DOCUMENTS has a two-level attribute structure.

describing the components used by a given component can be retrieved via the **uses** attribute.

Using virtual document classes *views* can be defined on the set of documents.

Because scanning a large library may be prohibitively expensive tools have been developed that support an efficient search (see Sect. 3.4).

Libraries The SEL supports different user classes, i.e. users, experts, project leaders, and a system administrator, each having different access privileges for the documents stored in the system. Documents, i.e. component descriptions, are organized in libraries at the following three levels:

1. User libraries belong to single users who have exclusive read and write access to the documents stored. New documents or document classes are initially stored in user libraries.
2. Project libraries contain project specific documents and are attached to single projects. Only the project members have reading access to this information. The project library is administered by the leader of the respective project who has read and write access.
3. Documents in the global library can be read by all SEL users. Documents of this library contain project overlapping information so that components developed for one specific project can be used in a number of different other projects. The system administrator has read and write access to the global library.

Documents and components can migrate from one library into another, e.g. to release and distribute a component or to submit a component for testing to a different software engineer. The quality of components and their descriptions that migrate to a "higher" library has to be assessed prior to migration. Quality assessment, however, is currently not directly covered by the SEL system.

3.3 The SEL as a Hypertext-system

The SEL System has been designed and implemented as a hypertext system. Component descriptions, i.e. documents, form the nodes of the hypertext network. Hyperlinks can be defined between any two documents that are related in some way. It should be emphasized that we do not restrict the type of the relationships in any respect and that the user is entirely free to decide what documents shall be linked. In general, links are defined at the instance level, i.e. between two individual documents. However, as will be seen later (Sect. 3.4) some relationships can be established independently from the particular document regarding only the document structure as defined in the corresponding document class, e.g. the "**uses-used**" relationship.

Links are typed, e.g. "**uses-used**" or "**imports-imported**" etc. In addition to conventional hypertext systems the SEL system also supports dynamic features (see Sect. 3.2 above). A sample hypertext document network is shown in Fig. 4.

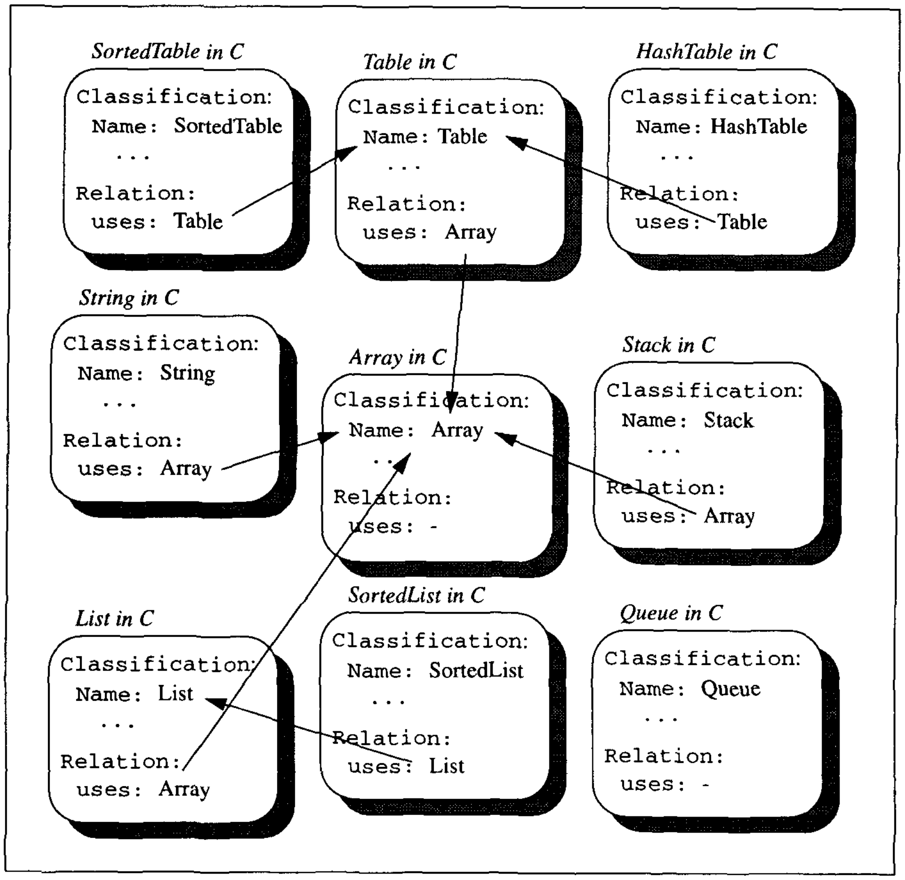

Fig. 4. Sample Hypertext Network taken from the BMW Application

3.4 Link Generation

The system supports the user in defining typed *links* between any two documents. Links that represent the same relationship R on the set of documents can be grouped. Thus links of type R materialize application specific and even user specific search heuristics that define a "R-neighbourhood" of a document. Normally, a document, and thus the component it describes, is selected whenever some user supplied selection condition is exactly matched by the component's properties. However, scanning a large library may be very costly. Links can be used to create an efficient search structure according to the specified condition.

Typed links can also be used to define *views* on a library, e.g. a view containing the documents related to a particular subproject.

Using links, the search for a document can optionally be extended to neighbouring documents[6]. In particular, this feature allows us also to relax the naming conventions that would otherwise have to obeyed in order to guide the search for a particular group of documents.

Let us first have a closer look at hyperlinks: A link consists of a *link anchor*, a *link destination*, and a *link type*. The link anchor structure can be formally described by a triple

$$(< document >, < attribute >, < text\ string\ within\ attribute\ value >).$$

The link destination corresponds to a tupel

$$(< document >, < attribute >).$$

The link type of a SEL-link can be decribed as a label defining the kind of the relation expressed by the hyperlink.

Semi-automatic link generation One of the features of the SEL system is the semi-automatic link generator. By defining a set of (pre-)conditions concerning the documents to be connected, the SEL link generator scans the document library and automatically creates hyperlinks between matching documents.

The following example explains the automatic link generation. Assume that a SEL user has created several documents, i.e. component descriptions, as instances of the document class MY_SIMPLE_DOCUMENTS. Let the documents be descriptions of implemented (procedures and) functions manipulating the data structures SORTED_TABLE, ARRAY, HASHTABLE, STACK etc. some of which are related via the "uses" attribute (see Fig. 4). The corresponding hyperlinks can be established semi-automatically as shown below.

1. Specification of the link anchor, i.e. the `document class` and `attribute` fields:
   `document class` := "MY_SIMPLE_DOCUMENTS"
   `attribute` := "`uses`"
2. Specification of the link destination:
   `document class` := "MY_SIMPLE_DOCUMENTS"
   `attribute` := "`Name`"
3. Specification of the link type, e.g.
   `Link-Type` := "`uses-used`"

The link generator is now able to create `uses-used`-hyperlinks between suitable documents belonging to the document class MY_SIMPLE_DOCUMENTS. Notice that it is not necessary to specify < *text string within attribute value* > when automatic link generation is performed. Instead, the tool automatically determines pairs of documents one (the origin) having an attribute `RELATION.calls`

[6] Practical experience has shown that the search should not go beyond the nearest neighbors because otherwise the search would result in a large set of only loosely connected documents.

the other (the destination) having an attribute **CLASSIFICATION.Name** and both containing the same text string in their attribute values.

The automatic link generator is an effective tool supporting the creation of the document network. However, there may exist other relationships between documents which cannot be generated by a pure inspection of the description. Therefore the system provides additional search tools, i.e. general queries, link lists, neighbourhood search, and link pattern search.

Queries The user can define a set of search keywords which describe the desired documents. Of course, keywords can be connected by the usual connectives **and**, **or**, and **not**. The search can be extended to synonyms by a *thesaurus*. The retrieved documents are sorted according to the hit rate corresponding to the keywords. As usual in the area of information retrieval it is assumed that there is a correlation between the hit rate and the relevance of a document. Though flexible general queries suffer from some drawbacks as regards efficiency and precision.

Link list This search tool exploits results of quotation analysis. It is based on the idea that the degree of importance of a document is proportional to the number of quotations. The hyperlinks correspond to the quotations with the consequence that a document which is often defined as a link destination is of great importance with respect to the relation represented by the hyperlink type.

Assume, for example, that a SEL user intends to find descriptions of the most frequently used components. Applying the SEL link list generator to the link type **uses-used** will return a list of these descriptions.

Practical experience has shown that the link list generator complements the other search tools in a useful way.

Neighbourhood search The documents that are connected to a given document d by links of type R originating from d define the *R-neighbourhood* of d. By the neighbourhood search tool the set D of documents that form the R-neighbourhood of d can be retrieved. The search for R-related documents is easy to understand. In field tests the neighbourhood search tool turned out to be the most effective tool of the SEL system.

Link pattern search This tool supports the search for "similar" documents according to the similarity measure described in the following. Quotation analysis has shown that the similarity of two documents is strongly related to the similarity of their quotation pattern [8]. A quotation of a document d corresponds to a link that points to d. The link pattern search tool determines for a given link type and a given document a list of similar documents that is ordered according to increasing similarity values. The latter are determined by a method developed by G. Salton [8] that is known to be effective in information retrieval. A drawback of this tool is its computational complexity. It is therefore best suited for static networks because in this case the similarity values can be pre-computed and stored.

4 Implementation

The data management component of the SEL system is based on the object-oriented database system GemStone. A full fledged graphical interface has been implemented using the Smalltalk-80 language features. Currently, the system is installed on SUN IPX-Workstations with 32 MBytes main memory and runs with very reasonable performance.

5 Other Work

While various systems have been developed that support the retrieval and configuration of software components from a library the idea to utilize hypertext links to guide the navigation through a network of software components seems to be entirely new. In particular, we did not find a system that exploits techniques borrowed from information retrieval to semi-automatically generate links between documents.

Garg and Scacchi [4, 5] present a hypertext-based system that models the entire software production process. In contrast, our system concentrates on the management of *existing components*. Consequently, we had to put an emphasis on the adaptability of document descriptions and on a sophisticated semi-automatic link generation. The Museion system [2] which is also hypertext-based provides features to manage and integrate all documents produced and used throughout the software life cycle. Documents can be classified using a thesaurus and facets[7]. However, the document structure is static and there is no link generation facility.

Batory and O'Malley report on the construction of hierarchical software systems from reusable components [1]. Their approach relies on some basic assumptions such as open architecture software and interface standardization which are frequently not met by existing component libraries. Hall [6] describes a generalization of executing each component of a library with test-inputs. This technique can be viewed as complementary to automated link generation. In very large libraries, however, the user may put emphasis on the efficiency of answer generation when he or she lets the system propose a set of suitable modules. Our system is able to support him or her in this respect. The SPADE system presented in [10] is a full fledged CASE tool that manages libraries of reusable software components and their descriptors. Different from the SPADE system, the navigation support offered by our system emphasizes semi-automatic link generation based on information retrieval techniques.

[7] The facet scheme was first proposed by Prieto-Diaz et al.. See e.g. [7].

6 Summary

A hypertext-based tool that supports the management of large libraries of reusable software components has been presented. A flexible classification of component descriptions as well as various means to interactively retrieve components that match a given specification are provided. Some novel approaches to the semi-automatic determination of relationships between component descriptions have been described and incorporated into the system that increase both the precision and the efficiency of component retrieval. This approach can also be applied when searching for hidden relationships in existing software libraries and thus qualifies our system for re-engineering tasks. In addition, concurrent multiuser access and several levels of access privileges are supported.

The SEL system described in this paper has been developed in cooperation with software engineers of BMW, Munich, Germany, and is currently installed at the BMW facilities in Munich. First field studies indicate that it has a good acceptance and is well-suited to the every-day reusability problems that have to be solved by software engineers. One direction of future work will be the integration of code modules and testing facilities.

Of course our simple, pragmatic approach also has its limitations. For instance, type correctness and interface correctness issues have not been addressed in this paper. We are therefore looking at generalizations, like e.g. adding expressive type disciplines, and allowing various means for semantic specifications (see [3, 11]). In addition we are planning to incorporate a faceted classification scheme [7] into the SEL system as a further improvement of its search facilities.

Acknowledgement

The author is indebted to K. Raith who was the project leader at BMW of the joint project. Thanks go also to K. Avini who implemented the SEL system.

References

1. D. Batory and S. O'Malley. The design and implementation of hierachical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology*, Oct. 1992.
2. M. Brorsson and I. Kruzela. Museion – a reuse support system for design of service features. In *Proc. 10th Annual International Phoenix Conference on Computers and Communications, 1991 Scottsdale, Arizona.* IEEE Computer Society Press, 1991.
3. B. Freitag, T. Margaria, and B. Steffen. A pragmatic approach to software synthesis. In *Proc. ACM SIGPLAN POPL'94 Post-Conference Workshop on Interface Definition Languages*, Portland, Oregon, Jan. 1994. (To Appear in ACM SIGPLAN Notices).
4. P. K. Garg and W. Scacchi. ISHYS - designing an intelligent software hypertext system. *IEEE Expert*, Fall 1989.

5. P. K. Garg and W. Scacchi. A hypertext system to manage software life-cycle documents. *IEEE Software*, May 1990.
6. R. J. Hall. Generalized behaviour-based retrieval. In *Proc. International Conference on Software Engineering.* IEEE Computer Society Press, 1993.
7. R. Prieto-Diaz and P. Freeman. Classifying software for reusability. *IEEE Software*, 18(1), Jan. 1987.
8. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw Hill, 1983.
9. J. Sametinger and A. Stritzinger. Exploratory software development with class libraries. In *Proc. 7th Joint Conference of the Austrian Computer Society, Klagenfurt, Austria*, 1992.
10. V. Seppänen, M. Heikkinen, and R. Lintulamp. SPADE – towards case tools that can guide design. In *Proc. Conference on Advanced Information Systems Engineering (CAISE '91), Trondheim, Norway*, 1991.
11. B. Steffen, T. Margaria, and B. Freitag. Module configuration by minimal model construction. Technical Report MIP-9313, Universität Passau, Passau, Germany, 1993.