SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 26(7), 815–832 (JULY 1996)

A Behavior-based Classification and Retrieval Technique for Object-oriented Specification Reuse

SHIH-CHIEN CHOU, JEN-YEN CHEN* AND CHYAN-GOEI CHUNG
Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu, Taiwan 30050, Republic of China
(email: jychen@csie.nctu.edu.tw)

SUMMARY

This paper presents a classification and retrieval technique for object-oriented specification reuse, based on the assumption that existing specifications exhibiting behaviors similar to that of the system under development are appropriate for reuse. Existing specifications are classified and retrieved according to the semantic networks abstracted from their behaviors. Since semantic networks attach semantic meanings to certain degrees of detail, our technique is rather precise. Primary behavior is used to classify specifications because it can be obtained in the early phases of system analysis. Therefore, our technique allows early reuse. Moreover, subspecifications and classes of existing specifications are classified independently so that they can be retrieved for reuse separately. Thus, our technique encourages reusing subspecifications as well as classes. Since a subspecification is composed of classes and their relationships, reusing it corresponds to reusing all those classes and relationships. A technique that reuses subspecifications as well as classes is thus expected to save more time than those that reuse only classes.

KEY WORDS: specification reuse; specification classification; specification retrieval; behavioral similarity

INTRODUCTION

Many software engineers agree that software reuse greatly improves software productivity and quality.[1–3] Software reuse is majorly based on these two approaches: generation and composition.[4] The latter is used in this paper, which composes existing software components to form new software systems.[5–8]

Many techniques for reusing program code have been developed.[5–12] To use these techniques, system specifications and designs must be available before program code can be reused. Thus, although implementation times are reduced, system analysis and design times are not. To enhance the power of software reuse, some researchers have developed techniques for reusing designs[13–18] or specifications.[19–22] Techniques for reusing specifications, for reusing designs, and for reusing program code can be integrated to support a reuse-based software development paradigm that can dramatically reduce software development time.

* Professor Chen is responsible for all communications.

CCC 0038–0644/96/070815–18

Received 9 January 1995
Revised 7 November 1995

Software reusability can be enhanced by the object-oriented (OO) approach because it has these merits: information hiding, modularity, abstraction, inheritance, and so on.[23] Reusability is thus one of the most important promises of the OO approach.[24–26] Considerable OO software reuse techniques have been developed.[9,10,17,25,27–28] However, few successful techniques for reusing OO specifications are available, which prompted us to design an OO specification reuse technique. This paper presents a classification and retrieval technique for OO specification reuse.

Software components can be classified and retrieved according to keywords,[29] facets,[30] attributes,[5,12] lexical affinities,[27,31] features,[32] or semantic networks.[33] Among them, the keyword approach is the simplest to implement and the semantic network approach is the most precise. Our technique is based on the semantic network approach, because specifications are semantically rich. Moreover, we combine this approach with the keyword approach for simplification purposes. In other words, a semantic network that classifies a specification is composed of keywords linked by various relationships.

PROCESS AND ISSUES OF OO SPECIFICATION REUSE

Basic considerations for our OO specification reuse technique are as follows:

1. Specifications that behave similarly are considered candidates for reuse. A specification primarily specifies functions and data.[34] Specifications with functions similar to those of the system under development are reusable. If the reused specification and the system under development have different data, data of the reused specification should be changed. Since the execution of functions exhibits behavior, specifications that behave similarly may have similar functions and hence may be reusable.
2. Classes as well as subspecifications, or even entire specifications, may be reused. Techniques that reuse only classes compose classes to form subsystems and then compose subsystems to form a system. To reduce reuse time, our technique reuses subspecifications and even entire specifications, in addition to classes. This reuse saves class composition time, because a specification or subspecification is composed of several classes and class relationships. (To enhance readability, in the remainder of this paper, a *subspecification* refers to either a subspecification or a complete specification.)

Before reuse, existing specifications must be classified and stored in a repository. They should be classified according to their behaviors so that those behave similarly to the system under development can easily be retrieved for reuse. To develop a system specification by reusing existing specifications, an analyst follows the OO specification reuse process below:

1. The analyst creates a *query* based on the behavior of the system under development. Since the analyst may not know the system's detailed behavior in this early phase of system analysis, the query should describe the system's *primary behavior* exhibited when its *primary functions* are executed.
2. A reuse support tool retrieves *candidate reusable subspecifications* that are existing subpsecifications with behaviors similar to that specified in the query.
3. The analyst examines the retrieved candidates and may select some for reuse. Those selected may need modification. After the reuse, subsystems of the system

under development whose behaviors are not primary may still be unspecified. The analyst should follow the above steps to deal with those subsystems.

4. If no candidates are retrieved in step (2) or none are selected for reuse, the analyst must decompose the system under development into subsystems and then follow the above steps to specify each subsystem. The rationale here is that subsystems whose behaviors are not primary may be able to reuse existing subspecifications.
5. The analyst specifies easy unspecified subsystems from scratch. Existing classes can be reused here if the analyst creates a query based on the intended class's object behavior and performs retrieval as described above. The retrieved classes should have object behaviors similar to that specified in the query, and are called *candidate reusable classes.*
6. The analyst composes the specified subsystem specifications and classes to form a complete specification.

According to the process outlined above, major issues for an OO specification reuse technique are: classification and retrieval of specifications, storage of specifications, examination of specifications, and modification and composition of specifications. This paper presents a technique for classification and retrieval of specifications.

CLASSIFICATION

Subspecifications and classes of existing specifications are independently classified so that they can be retrieved separately for reuse. Our technique does not physically put classified subspecifications or classes together. Instead, each subspecification or class is associated with a semantic network for classification purposes. This classification technique is thus a representation technique.[35]

Classification of classes

A class is classified by its object behavior, which can be represented by a state transition diagram.[23] An object changes states when one or more of its services (operations) are executed. Thus, executing a class's services demonstrates its object behavior. Class services can thus be used to represent class object behaviors.

A class service performs a primary operation on one or more attributes. The primary operation and attributes, which form a *service semantic network*, can be used to represent the service. For example, the service 'Borrow' of the class 'Book' performs the primary operation 'Borrow' on the attributes 'Identifier' and 'Status'. That service can be represented by the first service semantic network shown in Figure 1(a), in which the upper node denotes the service's primary operation, the lower nodes denote the attributes, and the links between them are 'Operate_on' relationships.

Since a class's services can be used to represent its object behavior, a class can be classified according to its services. Service semantic networks of a class's services can thus be combined to form a *class semantic network* for classifying the class. For example, the service semantic networks for the services of the class 'Book' (see Figure 1(a)) can be combined to form the class semantic network for that class (see Figure 1(b)). In a class semantic network, each node is represented by a keyword. Each keyword can have aliases to improve retrieval recall.[36] In a repository, the class semantic network as shown in Figure 1(b) is represented by the language description as shown in

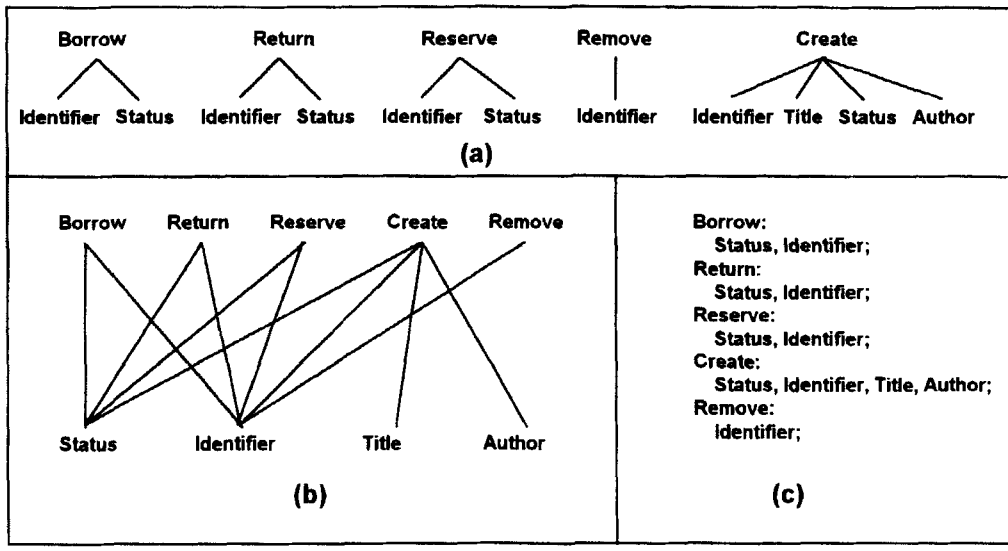

Figure 1. Service semantic network and class semantic network: (a) service semantic networks for the services of the class 'Book'; (b) notation for class semantic network of the class 'Book'; (c) language description for class semantic network in (b)

Figure 1(c). Thus, Figures 1(b) and 1(c) are considered to be interchangeable in this paper.

Classification of subspecifications

A subspecification is classified by the behavior exhibited when its functions are executed. According to the reuse process outlined above, subspecifications are usually retrieved for reuse in the early phases of system analysis, where a query describes the primary behavior of the system under development. To facilitate retrieval, subspecifications should be classified according to the primary behaviors exhibited when their primary functions are executed.

Executing a primary function triggers objects to accomplish the function. Since an object is triggered by invoking its class service, a subspecification's primary functions are accomplished by invoking some of its classes for service. These classes are *primary classes* that can be used to classify the subspecification. A primary class has several services that represent its object behavior. Invocation relationships exist among the primary classes that link their object behaviors to form a joint behavior for representing the subspecification's primary behavior. A subspecification can thus be classified by its primary classes, the services of those primary classes, and the invocation relationships among the primary classes. They form a *subspecification semantic network* as shown in Figure 2(a), where each node is represented by a keyword. In a repository, the network in Figure 2(a) is represented by the language description in Figure 2(b). Thus, Figures 2(a) and 2(b) are considered to be interchangeable in this paper.

A classification example

A simplified library system specification represented in Coad's model[37] is used as a classification example here. Its functional requirements are described briefly below.

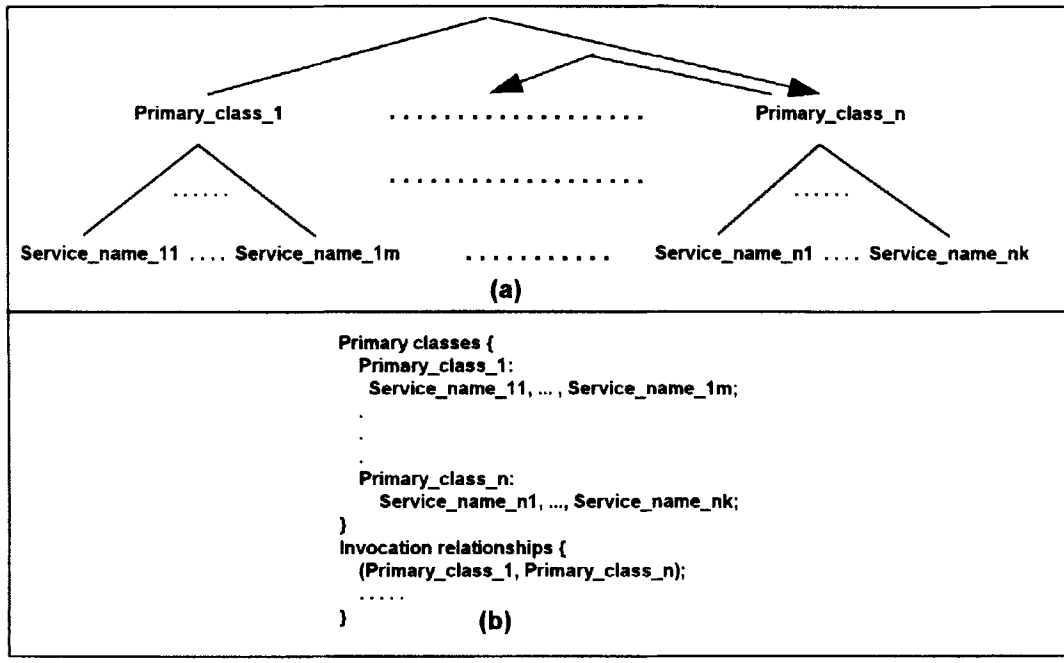

Figure 2. Notation and language description for subspecification semantic network: (a) notation; (b) language description

A library system should manage book and borrower status. Books can be borrowed by borrowers. Borrowed books can be returned. Books can be reserved. When a borrower borrows books, the amount he or she has borrowed should be increased by the number just borrowed. That amount should be decreased when the borrower returns books. New books can be added to the library, and obsolete books can be discarded.

A borrower's borrowing right can be suspended. A suspended right can be resumed. New borrowers can be added and current borrowers can be removed.

This system's specification is outlined in Figure 3, where two classes, 'Book' and

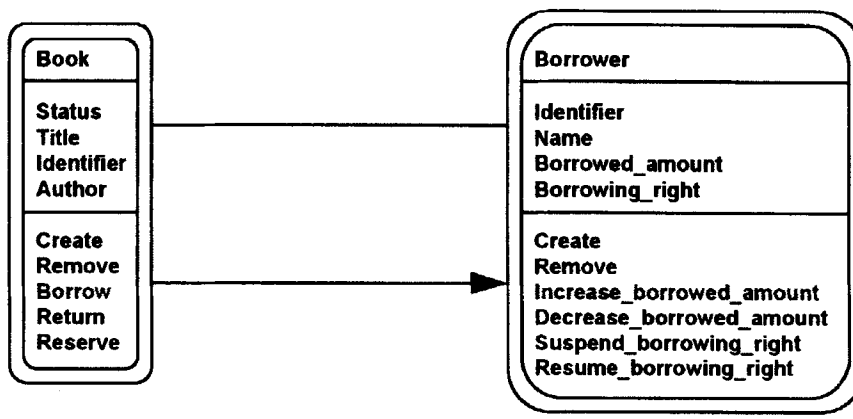

Figure 3. Specification outline for a simplified library system

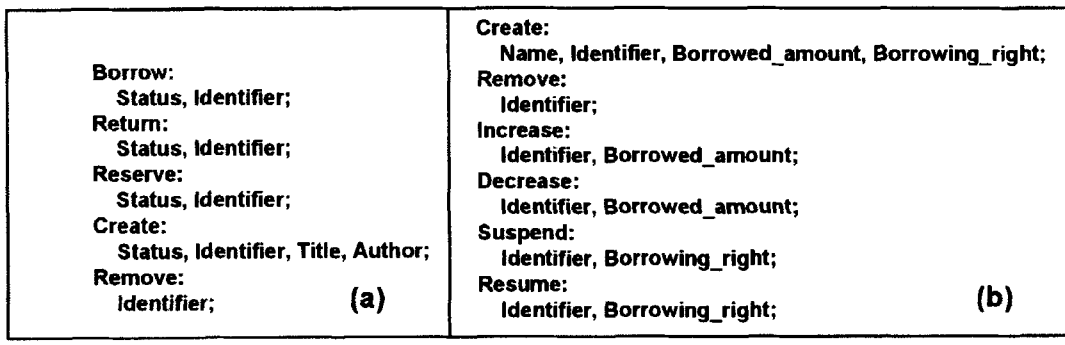

Figure 4. Class semantic networks for the simplified library system: (a) class semantic network for the class 'Book'; (b) class semantic network for the class 'Borrower'

'Borrower', are specified. An instance connection relationship between them connects a borrower and the books he or she borrows. Moreover, there is a message connection relationship between the two classes, because the services 'Borrow' and 'Return' of the class 'Book' invoke, respectively, the services 'Increase_borrowed_amount' and 'Decrease_borrowed_amount' of the class 'Borrower'.

Figure 4 shows the class semantic networks for the two classes. To classify the subspecifications, the specification is partitioned into these two subspecifications: 'Book management' and 'Borrower management'. Figure 5 shows the subspecification semantic networks for the specification and its subspecifications. For example, Figure 5(a) shows the semantic network for the subspecification 'Book management'

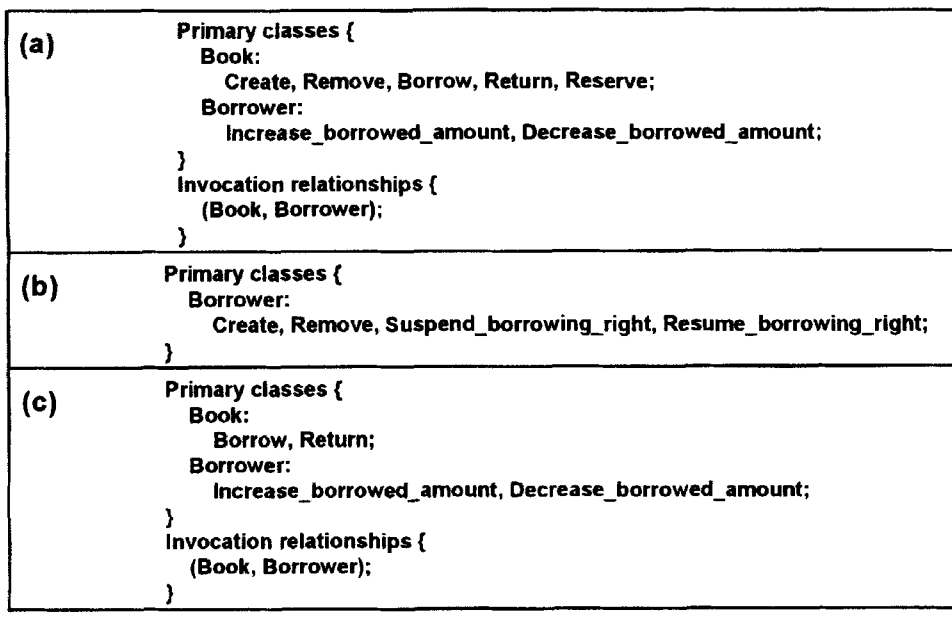

Figure 5. Subspecification semantic networks for the simplified library system: (a) the network for the subspecification 'Book management'; (b) the network for the subspecification 'Borrower management'; (c) the network for the entire specification

```
Primary classes {
  Tool:
    Create, Remove, Borrow, Return, Sell;
  Borrower:
    Increase_borrowed_amount, Decrease_borrowed_amount;
}
Invocation relationships {
  (Tool, Borrower);
}
```

Figure 6. Subsystem query for the subsystem 'Tool management'

whose primary functions are 'Borrow books', 'Return books', and so on. These primary functions are accomplished by invoking services of the classes 'Book' and 'Borrower', which are the primary classes of the subspecification. Since services of the class 'Book' invoke services of the class 'Borrower', there is an invocation relationship between the two primary classes.

RETRIEVAL

In retrieval, a query is created first. A query for retrieving candidate reusable subspecifications is a *subsystem query*, whereas that for retrieving candidate reusable classes is a *class query*. A subsystem query is to be compared with subspecification semantic networks in Figure 2(b). The format for this query should thus be the same as that shown in the Figure. For example, Figure 6 shows a subsystem query for the subsystem 'Tool management'. The format for a class query should be the same as the class semantic network in Figure 1(c). For example, Figure 7 shows a class query for the class 'Tool' in a toolroom.

A query is compared with subspecification semantic networks (or class semantic networks) in retrieval. The comparison results are used to compute similarities between the query and semantic networks. Such similarities are called *behavioral similarities* because both the query and the semantic networks describe behaviors. The greater the behavioral similarities are, the more reusable the subspecifications may be.

Behavioral similarity between a subsystem query and a subspecification semantic network

Matching classes and *matching invocation relationships* between a subsystem query and a subspecification semantic network are obtained by comparing the query with the network. A pair of matching classes consist of a class in the query and a class in the semantic network that have similar object behaviors. Since a class's object behavior is

```
Borrow:
  Status, Type;
Return:
  Status, Type;
Sell:
  Status, Type, Price;
Create:
  Status, Type, Price;
Remove:
  Type;
```

Figure 7. Class query for the class 'Tool'

represented by its services here, classes that possess services with the same names are considered matched. For example, the class 'Tool' in Figure 6 and the class 'Book' in Figure 5(a) are matched. A pair of matching invocation relationships consists of an invocation relationship in the query and that in the semantic network whose invoking classes are a pair of matching classes and whose invoked classes are also paired.

The comparison results can be represented by a Venn diagram, as shown in Figure 8. In the Figure, Q and S denote the set of classes and invocation relationships in the query, and that in the semantic network, respectively. M denotes the set of matching classes and invocation relationships that indicates the degree of similarity between the subspecification and the subsystem. UQ and US denote the sets of unmatched classes and invocation relationships that indicate the degree of dissimilarity. The more elements there are in the set M, the more reusable the subspecification may be. Conversely, the more elements there are in the sets UQ and US, the less reusable the subspecification may be. The behavioral similarity ($Ssub$) between a subsystem query and a subspecification semantic network can thus be roughly defined as a Jaccard's coefficient[36] shown below.

$$Ssub = \frac{|Q \cap S|}{|Q \cup S|} \tag{1}$$

To obtain a more precise behavioral similarity between a query and a subspecification semantic network, the comparison results from primary classes and those from invocation relationships should be weighted differently, because classes and invocation relationships have different effects on system behavior. Accordingly, equation (1) is adjusted as follows:

$$Ssub = Wc \frac{\sum_{i=1}^{|Qc \cap Sc|} Ssc_i}{|Qc \cup Sc|} + Wr \frac{|Qr \cap Sr|}{|Qr \cup Sr|} \tag{2}$$

In equation (2), Qc and Sc denote the set of primary classes specified in the query, and that specified in the semantic network, respectively. Qr and Sr denote the set of invocation relationships specified in the query, and that specified in the semantic network, respectively. Wc and Wr denote the weight of primary classes and that of invocation relationships, respectively. Ssc_i denotes the similarity between the ith pair of matching classes.

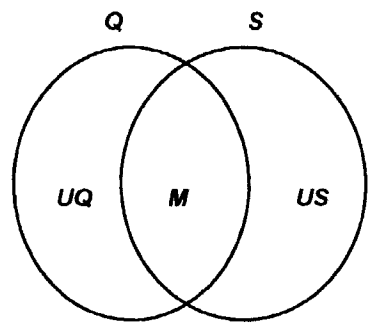

Figure 8. Venn diagram that illustrates comparison results

$$\frac{|Qr \cap Sr|}{|Qr \cup Sr|}$$

and

$$\frac{\sum_{i=1}^{|Qc \cap Sc|} Ssc_i}{|Qc \cup Sc|}$$

denote the Jaccard's coefficient for the invocation relationship comparison result and that for the class comparison result, respectively, where the latter's numerator is adjusted according to the similarities of matching classes. The similarity of each pair of matching classes is considered in equation (2) because the object behaviors of matching classes may be somewhat different. The similarity (*SSc*) between a pair of matching classes is defined below:

$$Ssc = \frac{|S1 \cap S2|}{|S1 \cup S2|} \tag{3}$$

where *S1* and *S2* denote the sets of services in the two classes, respectively.

Applying equations (2) and (3) to the query shown in Figure 6 and the semantic network shown in Figure 5(a) results in Table I.

Table I. Behavioral similarity between Figures 6 and 5(a)

Matching classes	Similarities of the matching classes	Matching invocation relationships	Behavioral similarity between the query and the semantic network
Tool and Book	2/3	Tool ⟶ Borrower and Book ⟶ Borrower	$Wc \times 5/6 + Wr$
Borrower and Borrower	1		

Note: *Wc* denotes the weight of classes. *Wr* denotes the weight of invocation relationships. A ⟶ B denotes the invocation relationship from class A to class B

Behavioral similarity between a class query and a class semantic network

Matching services between a class query and a class semantic network are obtained by comparing the query with the network. A pair of matching services consist of a service in the query and a service in the semantic network that have similar detailed operations. Since a service's primary operation is an abstraction of its detailed operations, services that have the same primary operation are considered matched. Based on the considerations employed in deriving equation (1), the behavioral similarity (*Scls*) between a class query and a class semantic network is defined below:

$$Scls = \frac{\sum_{i=1}^{|Qs \cap Ss|} Sser_i}{|Qs \cup Ss|} \tag{4}$$

where Qs and Ss denote the set of class services specified in the class query and that in the class semantic network, respectively, and $Sser_i$ denotes the similarity between the ith pair of matching services. Similarities of matching services are considered because they may have different detailed operations. Such a similarity ($Sser$) is defined below:

$$Sser = Wop + Watt \times Satt \tag{5}$$

where Wop and $Watt$ denote the weights of primary operations and attributes, respectively. $Watt$ is adjusted according to $Satt$, which is a number that indicates the similarity between the matching services' attributes. $Satt$ is defined below:

$$Satt = \frac{|A1 \cap A2|}{|A1 \cup A2|} \tag{6}$$

where $A1$ and $A2$ denote the sets of attributes in the two services, respectively.

Applying equations (4), (5), and (6) to the class query shown in Figure 7 and the class semantic network shown in Figure 4(a) results in Table II.

ENVIRONMENT SUPPORT

A prototype environment that supports the proposed technique has been implemented on an IBM PC. It is composed of a repository and the following tools: a specification editor, a specification classifier, a specification retriever, and a specification browser.

1. *Specification editor.* The editor is used to edit specifications that will be classified

Table II. Behavioral similarity between Figures 7 and 4(a)

Matching services	*Satt* of the matching services	Similarities of the matching services	Behavioral similarity between the query and the semantic network
`Create` and `Create`	1/6	*Wop* + *Watt*/6	(*Wop* × 4 + *Watt* × 5/6)/6
`Remove` and `Remove`	0	*Wop*	
`Borrow` and `Borrow`	1/3	*Wop* + *Watt*/3	(Note: If *Wop* and *Watt* are 0·5, the similarity is about 0·40.)
`Return` and `Return`	1/3	*Wop* + *Watt*/3	

Note: *Satt* denotes the similarity of attributes between matching services. *Wop* denotes the weight of primary operations. *Watt* denotes the weight of attributes

and stored in the repository for reuse. Figure 9 shows the window for this editor, where the classes 'Book' and 'Borrower' in a library system specification are being edited.

2. *Specification classifier.* This tool is used to edit semantic networks for subspecifications and classes. Figure 10 shows the window for editing subspecification semantic networks where a semantic network that consists of two primary classes, 'Book' and 'Borrower', and their invocation relationship is being edited. The window for editing class semantic networks is similar to that shown in Figure 10.
3. *Repository.* The repository stores specifications and their semantic networks. It is indexed by keywords to facilitate specification retrieval. Figure 11 shows the structure of the repository. Three keyword indices are in the repository: one for class service names, another for service primary operations, and the other for attributes. During retrieval, keywords in the query are used to search the indices. Semantic networks that have the same keywords as those in the query are then obtained using index pointers. Finally, candidate reusable classes or subspecifications are retrieved using semantic network pointers.
4. *Specification retriever.* This tool is used to edit queries, retrieve candidate reusable classes or subspecifications, and display the retrieved candidates. Figure 12 shows the window for editing subsystem queries. A query consisting of two primary classes, 'Tool' and 'Borrower', and their invocation relationship, is being edited. The edited query is then used to retrieve candidate reusable subspecifications. Figure 13 shows the window that displays retrieved candidates. Each candidate is associated with its behavioral similarity. The window for creating class

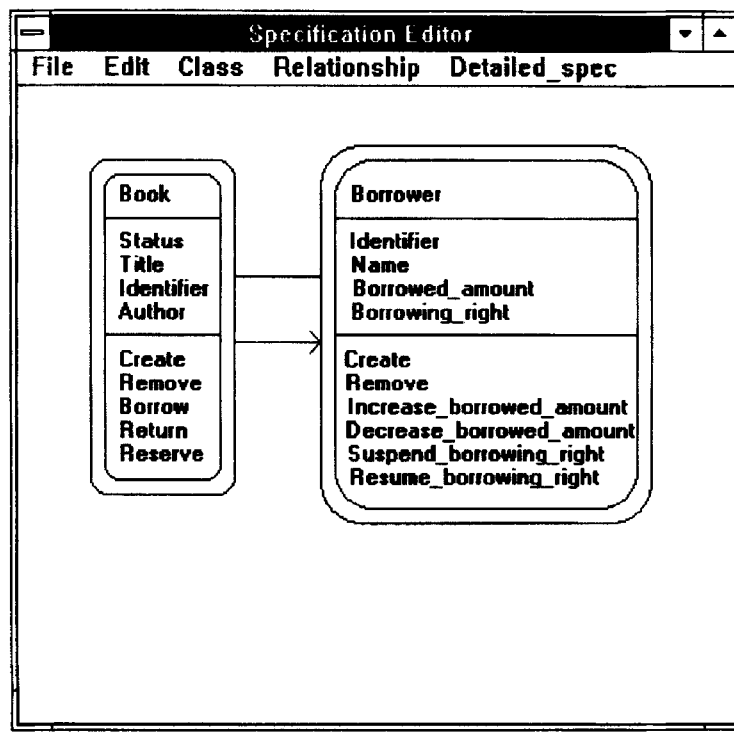

Figure 9. Window for specification editor

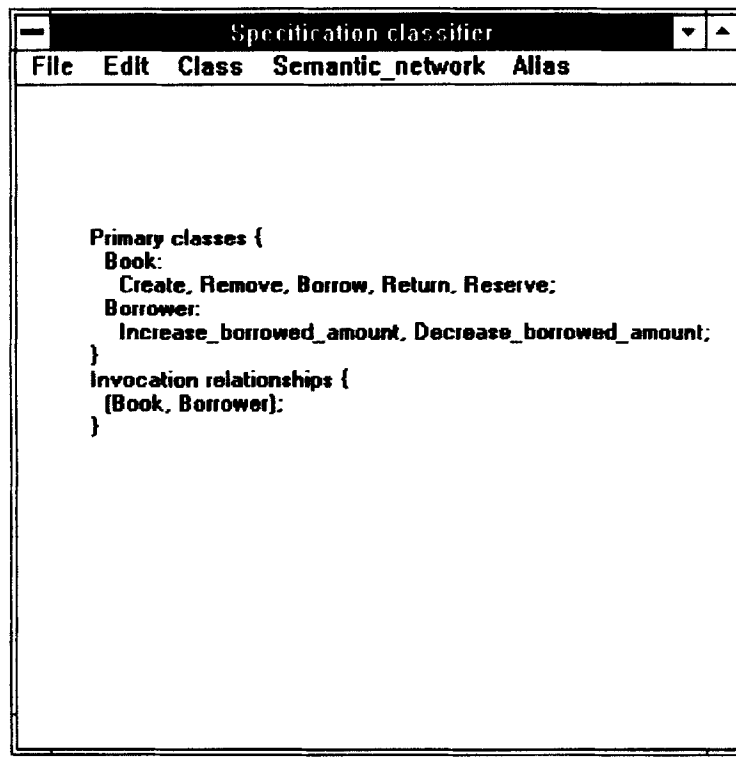

Figure 10. Window for classifying subspecifications

queries and that for displaying candidate reusable classes are similar to those shown in Figures 12 and 13, respectively.

5. *Specification browser.* This tool is used to browse through detailed specifications of the retrieved candidates. Figure 14 shows the specification browser window. A subspecification is outlined in the upper part of the window. Classes in the subspecification are listed in the lower right-hand corner. And the detailed specification of the selected class is displayed in the lower left-hand corner of the window.

EVALUATION

Our classification and retrieval technique is primarily for improving specification productivity in specification reuse. This improvement can be evaluated according to the following criteria:

1. *Large-scale reuse.* Reusing a large software component may save more time than reusing several smaller ones. Thus, large-scale reuse can save system analysis time.
2. *Early reuse.* System analysis is time-consuming. To reduce system analysis time, specification reuse should be done as early as possible.
3. *Specification retrieval precision and recall.* Specification retrieval should be precise. Otherwise, an analyst may spend much time to understand the retrieved subspecifications or classes that are not reusable. Moreover, retrieval recall should

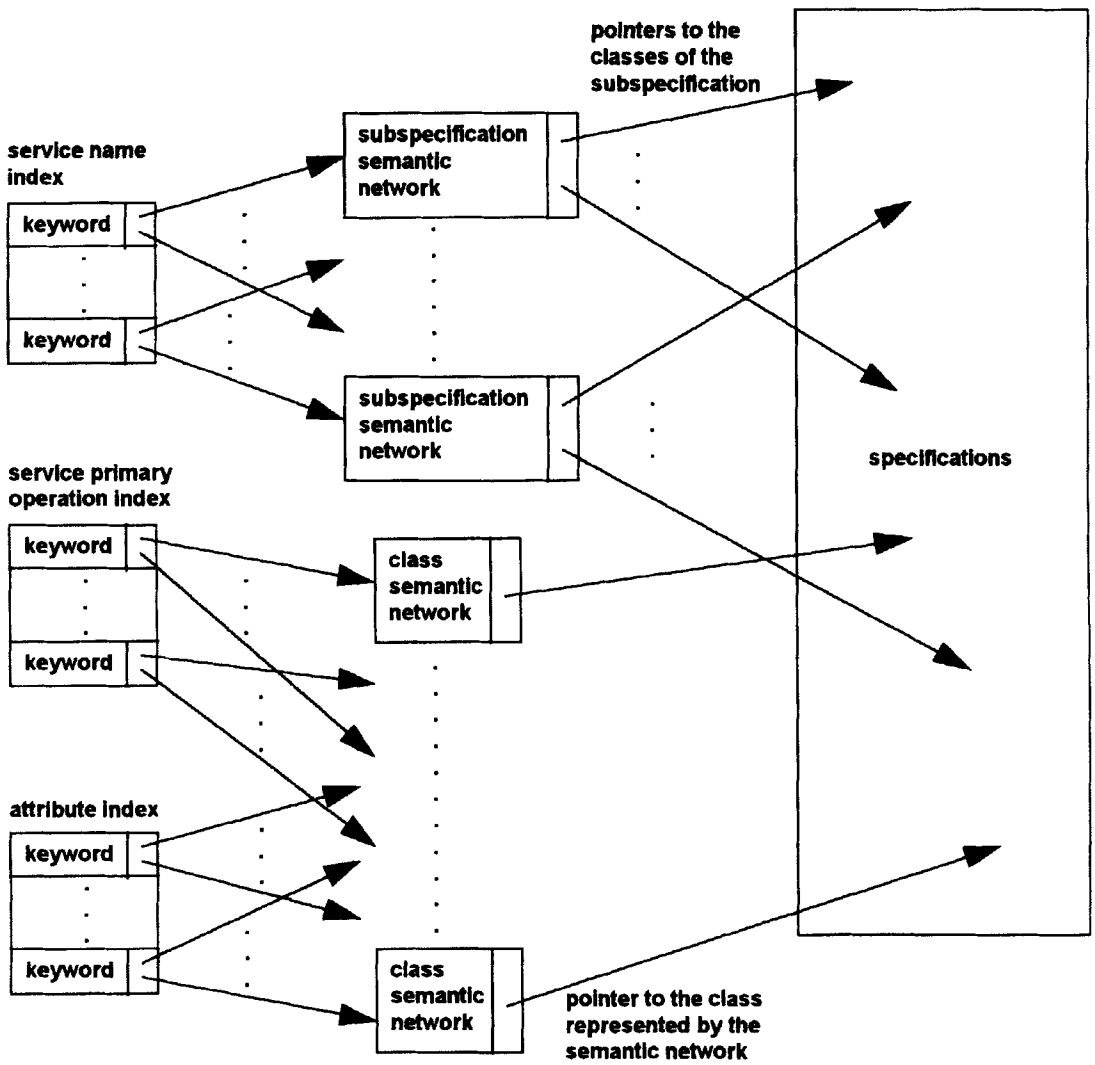

Figure 11. Structure of the repository

be high. Otherwise, an analyst may specify a specification from scratch, instead of reusing existing subspecifications or classes.

4. *Specification retrieval efficiency.* Retrieving candidate reusable specifications should be efficient. Otherwise, an analyst may spend much time waiting for retrievals.
5. *Reusable specification selection.* Guidance should be available to facilitate selection of appropriate candidate reusable specifications, because many candidates may be retrieved.

We will evaluate our technique according to the five criteria depicted above through experimentation. We will also evaluate reuse efficiency, which measures the reduction of system analysis time. For comparison purposes, four groups of experimenters will be involved. The first group will develop specifications from scratch. The second group

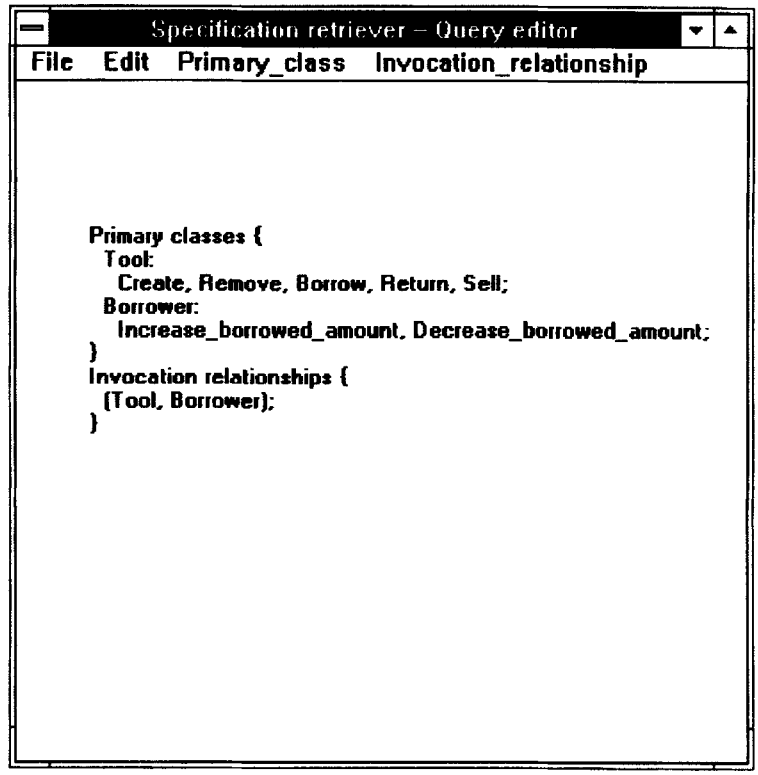

Figure 12. Window for editing subsystem query

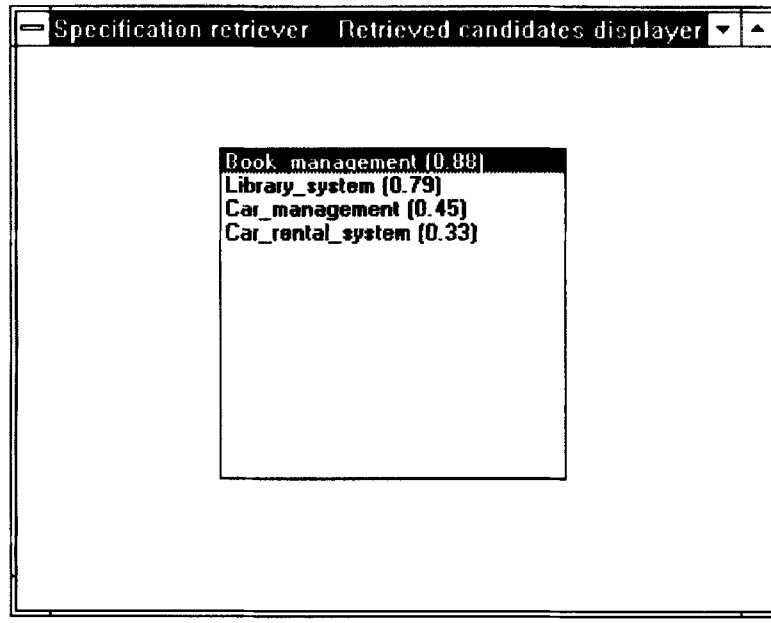

Figure 13. Window for displaying candidate reusable subspecifications

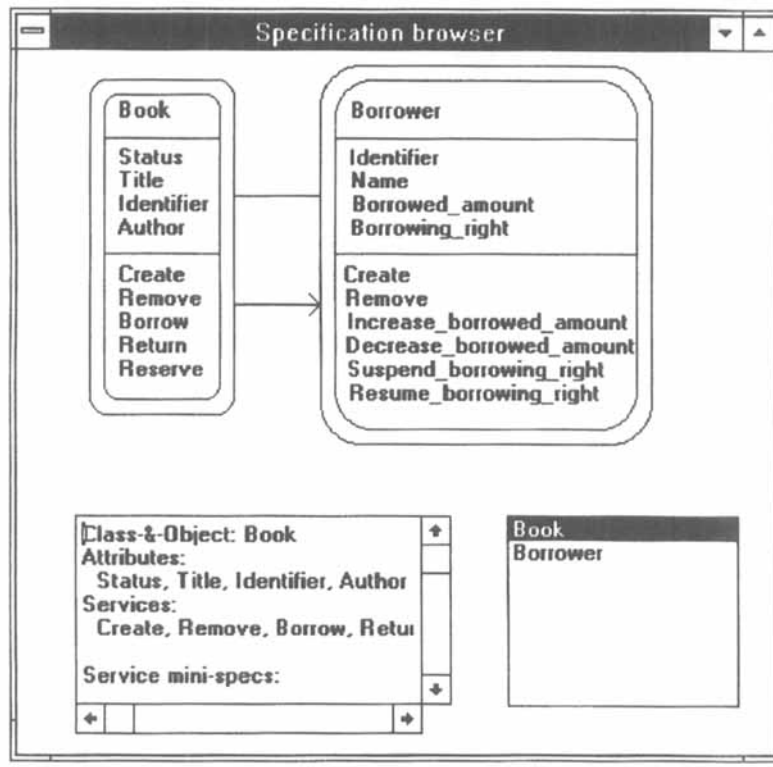

Figure 14. Window for specification browser

will apply a technique that reuses only classes in developing specifications. The third and fourth groups will apply techniques that reuse subspecifications as well as classes. They will apply, respectively, the well-known facet-based technique and our technique for specification classification and retrieval.

Before experimentation, several system specifications for a selected domain (e.g. management information systems) will first be developed. They will then be classified and stored in a repository.

Three experiments will be conducted to obtain the averaged performances of the assigned techniques under different situations. In the first experiment, the experimenters will develop specifications in the same domain as that of the existing specifications. In the second experiment, specifications to be developed are in a domain related to that of the existing specifications. In the third experiment, specifications to be developed are in an unrelated domain. The following data will be collected in the experiments:

1. *The number of classes in each reused subspecification.* This number will show the size of the reused specification. The larger the number is, the larger scale the reuse is.
2. *The time when the experimenters start to reuse existing specifications.* This data will be used to evaluate whether the assigned techniques encourage early reuse.
3. *The time needed to complete each specification retrieval.* This data will be used to evaluate retrieval efficiency.
4. *The number of candidate reusable subspecifications in each retrieval* (Nc), *and*

*the number of candidates with behaviors similar to that of the system under development (*Nr*).* These values will be used to evaluate retrieval precision and recall. The number *Nr/Nc* will indicate retrieval precision. The number *Nr/Nt* will indicate retrieval recall, where *Nt* is the number of subspecifications in the repository that have behaviors similar to that of the system under development.

5. *The behavioral similarity of each retrieved candidate.* This data will be used to evaluate whether behavioral similarities are good guides for selecting appropriate candidate reusable subspecifications. If the retrieved candidates with higher behavioral similarity values are more reusable, behavior similarities are good guides for the selection.
6. *The total time needed to develop each specification.* This data will be used to evaluate reuse efficiency.

The data collected from the three experiments will be averaged. The averaged values will show the performances of the techniques used in the experiments. We hope that our technique will out-perform the others.

To conduct the above experiments, a complete specification reuse support environment must be set up. In addition, the experiments are expected to take a long time. Therefore, the experimental results are not currently available. Nevertheless, an informal evaluation of our technique according to the above five criteria was carried out and the following results were obtained:

1. Our technique classifies subspecifications and classes independently so that they can be retrieved separately. It thus supports reusing subspecifications as well as classes, and thus encourages large-scale reuse.
2. Our technique classifies subspecifications according to their primary behaviors. Candidate reusable subspecifications can thus be retrieved according to the primary behavior of the system under development, which is available in the early phases of system analysis. Our technique thus encourages early reuse.
3. Our technique classifies and retrieves specifications based on their behaviors as represented by semantic networks. Since specifications that behave similarly are considered candidates for reuse, a behavior-based technique can improve retrieval precision. Moreover, since semantic networks attach semantic meanings to certain degrees of detail, the semantic network approach is rather precise. Accordingly, specification retrieval using our technique is expected to be precise.

   To increase retrieval recall, each keyword in semantic networks can have several aliases. Increasing recall, however, may decrease precision. This problem can be solved by examining the retrievals.
4. Specification retrieval using the semantic network approach is indeed inefficient. This inefficiency is thus a weakness of our technique. However, our technique improves retrieval efficiency by combining the semantic network approach with the keyword approach. Moreover, keyword indices are stored in the repository to further facilitate retrieval.
5. The reusability of existing specifications can be determined by their behavioral similarities to the query. Our technique quantifies such similarities into numeric values. The larger the values are, the more reusable the specifications may be. Behavioral similarities can thus be used as a guide to select appropriate candidates for reuse.

CONCLUSIONS AND FUTURE WORK

This paper presents a specification classification and retrieval technique based on the semantic network approach. Since semantic networks attach semantic meanings to certain degrees of detail, our technique seems to be rather precise. It classifies and retrieves specifications according to their primary behaviors, which are available in the early phases of system analysis. It thus allows early reuse. Moreover, our technique reuses subspecifications as well as classes; thus it reduces the time needed to compose classes. Based on its early reuse ability and reduction of class composition time, our technique is expected to improve reuse efficiency.

Currently, only a prototype environment has been developed for the proposed technique. In the future, we are going to complete the following work so that specification reuse can be further facilitated:

1. *Design an executable specification language to facilitate specification understanding.* Candidate reusable subspecifications and classes must be understood before they can be reused. A good approach to gaining this understanding is to execute the candidates. An executable specification language is thus needed.
2. *Enhance the specification editor function to facilitate specification modification and composition.* Modification is necessary when the reuse subspecifications do not exactly fit the system under development. Moreover, a specification may be composed of several reused subspecifications and classes. The specification editor should thus facilitate specification modification and composition.
3. *Construct a complete specification reuse support environment.* We hope to evaluate our reuse technique by means of practical experiments, and adjust it according to the experimental results. A complete reuse support environment is therefore necessary.

REFERENCES

1. T. Biggerstaff and C. Richter, 'Reusability framework, assessment, and directions', *IEEE Software*, **4**, 41–49 (1987).
2. V. R. Basili and H. D. Rombach, 'Support for comprehensive reuse', *Software Eng. J.*, **6**, 303–316 (1991).
3. W. C. Lim, 'Effects of reuse on quality, productivity, and economics', *IEEE Software*, **11**, 23–30 (1994).
4. T. J. Biggerstaff and A. J. Perlis, *Software Reusability, Vol. I, Concepts and Models*, Addison-Wesley, New York, 1989.
5. G. Fisher, S. Henninger and D. Redmiles, 'Cognitive tools for locating and comprehending software objects for reuse', *Proc. 13th International Conference on Software Engineering*, 1991, pp. 318–328.
6. B. A. Burton, R. W. Aragon, S. A. Bailey, K. D. Koehler and L. A. Mayes, 'The reusable software library', *IEEE Software*, **4**, 25–33 (1987).
7. M. Lenz, H. A. Schmid, and P. F. Wolf, 'Software reuse through building blocks', *IEEE Software*, **4**, 34–42 (1987).
8. D. J. Chen and P. J. Lee, 'On the study of software reuse using reusable C++ components', *J. Syst. Software*, **20**, 19–36 (1993).
9. B. M. Kennedy, 'Design for object-oriented reuse in the OATH library', *J. Object-Oriented Progr.*, **5**, 51–57 (1992).
10. P. Johnson and C. Rees, 'Reusability through fine-grain inheritance', *Software—Practical Experience*, **22**, 1049–1068 (1992).
11. A. Podgurski and L. Pierce, 'Retrieving reusable software by sampling behavior', *ACM Trans. Software Eng. Methodol.*, **2**, 286–303 (1993).
12. S. Henninger, 'Using iterative refinement to find reusable software', *IEEE Software*, **11**, 48–59 (1994).
13. M. D. Lubars, 'The IDeA design environment', *Proc. 11th International Conference on Software Engineering*, 1989, pp. 23–32.

14. M. D. Lubars and M. T. Harandi, 'Knowledge-based software design using design schemas', *Proc. 9th International Conference on Software Engineering*, 1987, pp. 253–262.
15. S. Katz, C. A. Richter and K.-S. The, 'PARIS: a system for reusing partially interpreted schemas', *Proc. 9th International Conference on Software Engineering*, 1987, pp. 377–385.
16. G. Arango, E. Schoen and R. Pettengill, 'A process for consolidating and reusing design knowledge', *Proc. 15th International Conference on Software Engineering* 1993, pp. 231–242.
17. D. J. Chen and D. T. K. Chen, 'An experimental study of using reusable software design frameworks to achieve software reuse', *J. Object-Oriented Progr.*, **7**, 56–67 (1994).
18. S. Khajenoori, D. G. Linton and C. A. Morris, 'Enhancing software reusability through effective use of the essential modelling approach', *Info. Software Technol.*, **36**, 495–501 (1994).
19. N. Maiden, 'Analogy as a paradigm for specification reuse', *Software Eng. J.*, **6**, 3–15 (1991).
20. A. Finkelstein, 'Re-use of formatted requirements specifications', *Software Eng. J.*, **3**, 186–197 (1988).
21. N. A. M. Maiden, 'Saving reuse from the noose: reuse of analogous specifications through human involvement in reuse process', *Info. Software Technol.*, **33**, 780–790 (1991).
22. N. A. Maiden and A. G. Sutcliffe, 'Exploiting reusable specifications through analogy', *Commu. ACM*, **35**, 55–64 (1992).
23. G. Booch, *Object Oriented Analysis and Design with Applications*, 2nd edn, The Benjamin/Cummings Publishing Company, Inc., 1994.
24. B. Meyer, *Object-Oriented Software Construction*, Prentice-Hall, New Jersey, 1988.
25. B. Meyer, 'Reusability: the case for object-oriented design', *IEEE Software*, **4**, 50–64 (1987).
26. J. A. Lewis, S. M. Henry, D. G. Kafura and R. S. Schulman, 'An empirical study of the object-oriented paradigm and software reuse', *Proc. OOPSAL'91*, 1991, pp. 184–196.
27. R. Helm and Y. S. Maarek, 'Integrating information retrieval and domain specific approaches for browsing and retrieval in object-oriented class libraries', *Proc. OOPSAL'91*, 1991, pp. 47–61.
28. P. Coad, 'Object-oriented patterns', *Commun. ACM*, **35**, 152–159 (1992).
29. W. B. Frakes and B. A. Nejmeh, 'Software reuse through information retrieval', *Proc. 20th HICSS*, 1987, pp. 530–535.
30. R. P. Diaz and P. Freeman, 'Classifying software for reusability', *IEEE Software*, **4**, 6–16 (1987).
31. Y. S. Maarek, D. M. Berry and G. E. Kaiser, 'An information retrieval approach for automatically constructing software libraries', *IEEE Trans. Software Eng.*, **SE-17**, 800–813 (1991).
32. E. Ostertag, J. Hendler, R. P. Diaz and C. Braun, 'Computing similarity in a reuse library system: an AI-based approach', *ACM Trans. Software Eng. Methodol.*, **1**, 205–228 (1992).
33. P. Devanbu, R. J. Brachman, P. G. Selfridge and B. W. Ballard, 'LaSSIE: a knowledge-based software information system', *Proc. 12th International Conference on Software Engineering*, 1990, pp. 249–261.
34. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 3rd edn, McGraw Hill, 1992.
35. W. B. Frakes and T. P. Pole, 'An empirical study of representation methods for reusable software components', *IEEE Trans. Software Eng.*, **SE-20**, 617–630 (1994).
36. C. J. van Rijsbergen, *Information Retrieval*, Butterworths-Heinemann, UK, 1979.
37. P. Coad and E. Yourdon, *Object-Oriented Analysis*, 2nd edn, Prentice-Hall, New Jersey, 1991.