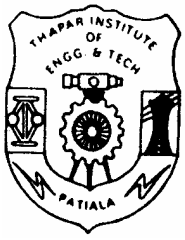
Retrieving Best Component From Reusable Repository

A thesis
submitted for the partial fulfillment of requirement
for the award of the degree
of

Master of Engineering
in
Software Engineering



Under the guidance of
Mr. Rajesh K. Bhatia
Asstt. Professor, CSED
TIET, Patiala

Submitted By
Navneet Kaur
(Roll No-8033112)

Computer Science & Engineering Department
Thapar Institute of Engineering and Technology,
(Deemed University),
Patiala (Punjab)-147004
June 2005

Abstract

The main purpose of information retrieval system is to retrieve the information according to user need. In principle, information storage and retrieval is simple. But practically the effective information retrieval is not as simple. Much of the research and development in information retrieval is aimed at improving the effectiveness and efficiency of retrieval. Effective component retrieval from a large repository is one such aim of the on going research. In software component repository thousand of components are stored using various classification techniques. Retrieving a software component that can be reused is a tedious process. Software components have certain attributes associated with them, and each attribute has relative importance for that component, which is called as weights. Retrieving the components considering these features becomes more difficult and time consuming. The objective of thesis is to select the best component from a repository that can be reused. In the proposed system one such component repository is developed. From repository best component has to be found. Best components are retrieved in a two step process. The first step gives all the relevant components, and the second step gives the best component to the user. This enhances the chances of retrieval of best component from a repository. The first technique used is simple keyword based retrieval and second technique is genetic algorithm. Genetic algorithms give satisfactory results for those components which have attributes and weights. The genetic algorithms based technique is very effective when the repository size is very large. It is based on mechanism of natural selection and natural genetics.

Declaration

I hereby certify that the work which is being presented in the thesis entitled "Retrieving Best Component from Reusable Repository" in the partial fulfillment of the requirements for the award of the degree of Master of Engineering (Software Engineering) of Thapar Institute of Engineering & Technology (Deemed University), Patiala, is an authentic record carried out under the supervision of Mr. Rajesh K. Bhatia. The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other university.

Navneet Kaur

This is to certify that the above statement made by the candidate is correct and true to my best of knowledge.

Mr. Rajesh K. Bhatia
Assistant Professor
Computer Sc. & Engg. Department
Thapar Institute of Engg. & Technology, Patiala

Countersigned by

Mr. R.S. Salaria
Assistant Professor & Head
Computer Sc. & Engg. Department
Thapar Institute of Engg. & Technology
Patiala

Dr. D.S. Bawa
Dean (Acedemic affairs)
Thapar Institute of Engg. & Technology
Patiala

Acknowledgement

The field of research is very vast. The contributions in one way or the other by the technocrats are very useful for the humanity at large who is to derive benefits. Sometime my mind goes to my wonderful dream of childhood to become an engineer and by the grace of god that dream is fulfilled now. Let me pay my thanks to the almighty god who showered kindness to achieve the goal for which I had high aspirations.

Really I am extremely grateful to my most esteemed guide Mr. Rajesh K. Bhatia, Asstt. Professor of Computer Science and Engineering Department for his able guidance, first hand experience, motivation and sympathetic attitude to write and complete my thesis. He has qualities to be an ideal guide. The way he helped me in my thesis work, requires no elaboration. Hence he will always prevail upon my remembrance.

I may not forget to pay my special thanks to Mr. R.S. Salaria, Head of Computer Science and Engineering Department, who infused confidence in me to complete my studies. He has been a source of great motivation and encouragement throughout my studies and even writing thesis. It is praiseworthy and admirable.

I have loving parents who have been very nice to me. They always helped me not only with the financial support but also with moral support. I am also indebted to my respected grandparents and my elder brother who acted as a source of inspiration. Last but not least I pay my thanks to other staff members for their help from time to time.

All may not be mentioned but none is forgotten.

Table of Contents

List of Figures

List of Tables

Tables **Page No**

Organization of Thesis

In the thesis, Chapter 1, include Component based software engineering, its process, reusable Software components and their characteristics, it also includes design for reuse.

Chapter 2 gives the overview of information retrieval process, and its various classification and retrieval techniques. Pros and cons of these techniques are discussed. Soft computing techniques are also discussed.

Chapter 3 includes proposed model for information retrieval. Proposed model detail, component descriptor, and its design. Keyword based retrieval and genetic algorithms approach for optimization is included.

Chapter 4 contains conclusion, case study of the proposed model, and future scope.

Chapter 1
Component Based Software Engineering

Change is a badge of modern corporate. No corporate can do business without software. Now, change is a badge of software community. We are witnessing a historical encounter of change of environment and change of software technology. Widespread use of the personal computers and the Internet make computers commodity goods and created new market and users. This requires substantial change to software industry. New users, most of them are consumers; require to drastically reduce the cost of software in order to match the ever-decreasing hardware price. Demands to new applications such as electronic commerce and groupware are high. However, conventional methodologies have not achieved such drastic gain of the productivity and quality yet. The use of components is the primary source of the productivity and quality.

Making applications from software components has been a dream in software engineering community since its very early time. As quoted in the literatures, McIlroy

Wrote in the NATO conference in 1968; "*My thesis is that the software industry is weakly founded, in part because of the absence of a software components sub industry. ..... A components industry could be immensely successful*". However, wide spread reuse of software components over the industry has not come true. A number of obstacles have been identified. However, it seems clear that the most fundamental problems are lack of mechanisms to make components interoperable and lack of "really reusable" components. Component based software engineering has promised to revolutionize the computer industry. It has the potential to enable Software developers to quickly and economically construct meaningful programs from a toolkit of existing components.

Since early 1990's, so-called Component-Based Software Engineering (CBSE) or componentware have emerged. At the early time, CBSE emphasized on the *EUC (End-User Computing)* such as composing applications on the PCs. However, the use of *COTS (Commercial Off-The-Shelf)* software promoted the CBSE in the development business applications. Furthermore, quick evolution of the Internet technology such as Web and Java-based technologies even open up new possibilities of CBSE such as network distribution of components, and the reuse and interoperation of components over the Internet. Now, a few set of technologies have been widely deployed and are still evolving. They include ActiveX/DCOM from Microsoft, CORBA from OMG and JavaBeans from SUN Microsystems.

1.1 WHAT IS CBSE?

"CBSE is a process that aims to design and construct software systems using reusable software components". If one is familiar with Object oriented programming (OOP), it can be useful to think of CBSE in a similar way. In OOP, code is reused in the form of objects. These objects are often contained in

vast libraries of reusable code. Frameworks take the process even further, providing more robust and disciplined systems of reuse. By obtaining and reusing parts of systems which have already been 'tried and tested', we can exploit the principal advantages of object-oriented programming techniques over procedural programming techniques, which is that they enable programmers to create modules1 that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify. In the same way, in CBSE, by reusing an existing component you cut out a lot of the hard work with establishing the usefulness and in testing that component – although, as we will see, some testing will still be required [18,15].

Software reuse is one of the main motivations for CBSE. It focuses on reusing and adapting existing components instead of developing them from scratch. When a large software system is decomposed into smaller, independent components it is usually possible to reuse these in later systems. This reduces both the development cost and the cost associated with quality and integrity and increases the return from investments [2].

1.2 WHY CBSE?

The goal of component-based software engineering is to increase the productivity, quality, and time-to-market in software development thanks to the deployment of both standard components and production automation. One important paradigm shift implied here is to build software systems from standard components rather than "reinventing the wheel" each time. This requires thinking in terms of system families rather than single systems. CBSE uses Software Engineering principles to apply the same idea as OOP to the whole process of designing and constructing software systems. It focuses on *reusing* and *adapting* existing components, as opposed to just coding in a particular style. CBSE encourages the composition of software systems, as opposed to programming them [15].

Two important aspects of the question "why CBSE now?" are:

First, several underlying technologies have matured that permit building components and assembling applications from sets of those components. Object oriented and Component technologies are examples – especially standards such as CORBA.

Second, the business and organizational context within which applications are developed, deployed, and maintained has changed. There is an increasing need to communicate with legacy systems, as well as constantly updating current systems. This need for new functionality in current applications requires technology that will allow easy additions.

CBSE should, in theory, allow software systems to be more easily assembled, and less costly to build. Although this cannot be guaranteed, the limited experience of adopting this strategy has shown it to be true. The software systems built using CBSE are not only simpler to bold and cheaper – but usually turn out to be more robust, adaptable and updateable. CBSE allows use of predictable architectural patterns and standard software architecture leading to a higher quality end result.

1.3 WHAT IS A COMPONENT?

Brown and Wallnau describe a component as *"a nontrivial, nearly independent, and replaceable part of a system that fulfils a clear function in the context of a well defined architecture"*.

Brown and Wallnau describe a Software component as *"a unit of composition with contractually specified and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties"*.

Another Definition of Software component is also described as "s*oftware components are binary units of independent production, acquisition, and deployment that interact to form a functioning system"*

1.4 CHARACTERISTICS OF COMPONENT

The characteristics of a component are:

- It is a marketable entity. A component is a self-contained, shrink-wrapped, binary piece of software that one can typically purchase in the open market.
- It is not a complete application. A component can be combined with other components to form a complete application. It is designed to perform a limited set of tasks within an application domain.
- It can be used in unpredictable combinations. Like real-world objects, a component can be used in ways that were totally unanticipated by the original developer. Typically, components can be combined with other components of the same family – called suites using plug-and-play.
- It has a well-specified interface. Like a classical object, a component can only be manipulated through its interface. This is how the component exposes its function to the outside world. A component's interface is separate from its implementation. A component can be implemented using objects, procedural code, or by encapsulating existing code.
- It is an interoperable object. A component can be invoked as an object across address spaces, networks, languages, operating systems, and tools. It is a system-independent software entity.
- It is an extended object.

Thus a component is a reusable, self-contained piece of software that is independent of any application.

1.5 REUSABLE SOFTWARE COMPONENTS

The main Reusable software components are [3, 20, 24]:

***Project Plans*:** The basic structure and much of the content (e.g., the SQA plan) of a software project plan) can be reused from project to project. This reduces the time required to develop the plan. Schedules,

budgets, assignments, and status evaluations for the management of an Application Engineering project are parts of project plan.

Cost Estimates: Because similar function is often implemented in different projects, it may be possible to reuse to estimates for that function with little or no modification.

Architecture: Reuse of architectures include component interface, interconnections between components, and platform configuration.

Requirements: Models and specifications for classes and objects are obvious candidates for reuse, In addition, analysis models (e.g., data flow diagrams) developed using conventional software engineering approaches can also be reused.

Designs: Architectural, data interface, and procedural designs developed using conventional methods are candidates for reuse. More commonly, system and object designs are reused.

***Source Code*:** Verified program components written in compatible programming languages are candidates for reuse.

User and Technical Documentation: It is often possible to reuse large portions of user and technical documentation, even though the specific applications differ.

Human interfaces: Probably the most widely reused software artifact, GUI software is commonly reused. Because it can account for 60 percent of the code volume of an application, the benefits of reuse are significant.

Data: Among the most commonly reused artifacts, data encompasses internal tables, lists, and record structures, as well as files and complete databases.

Test cases: Whenever a design or code component is to be reused, the relevant test case should be "attached" to it.

1.6 THE CBSE PROCESS

CBSE is in many ways similar to conventional or object-oriented software engineering. A software team establishes requirements for the system to be built using conventional requirements elicitation techniques. An architectural design is established. Here though, the process differs. Rather than a more detailed design task, the team now examines the requirements to determine what subset is directly amenable to *composition*, rather than *construction* [15, 22].

For each requirement, the team will ask:

- Are commercial off-the-shelf (COTS) components available to implement the requirement?
- Are internally developed reusable components available to implement the requirement?
- Are the interfaces for available components compatible within the architecture of the system to be built?

The team will attempt to modify or remove those system requirements that cannot be implemented with COTS or in-house components. This is not always possible or practical, but reduces the overall system cost

and improves the time to market of the software system. It can often be useful to prioritize the requirements, or else developers may find themselves coding components that are no longer necessary as they have been eliminated from the requirements already.

The CBSE process identifies not only candidate components but also qualifies each components' interface, adapts components to remove architectural mismatches, assembles components into selected architectural style, and updates components as requirements for the system change.

CBSE has two major activities: *Domain Engineering*, which produces components for future reuse, and *Component-Based Development*, which produces applications that reuse the existing components.

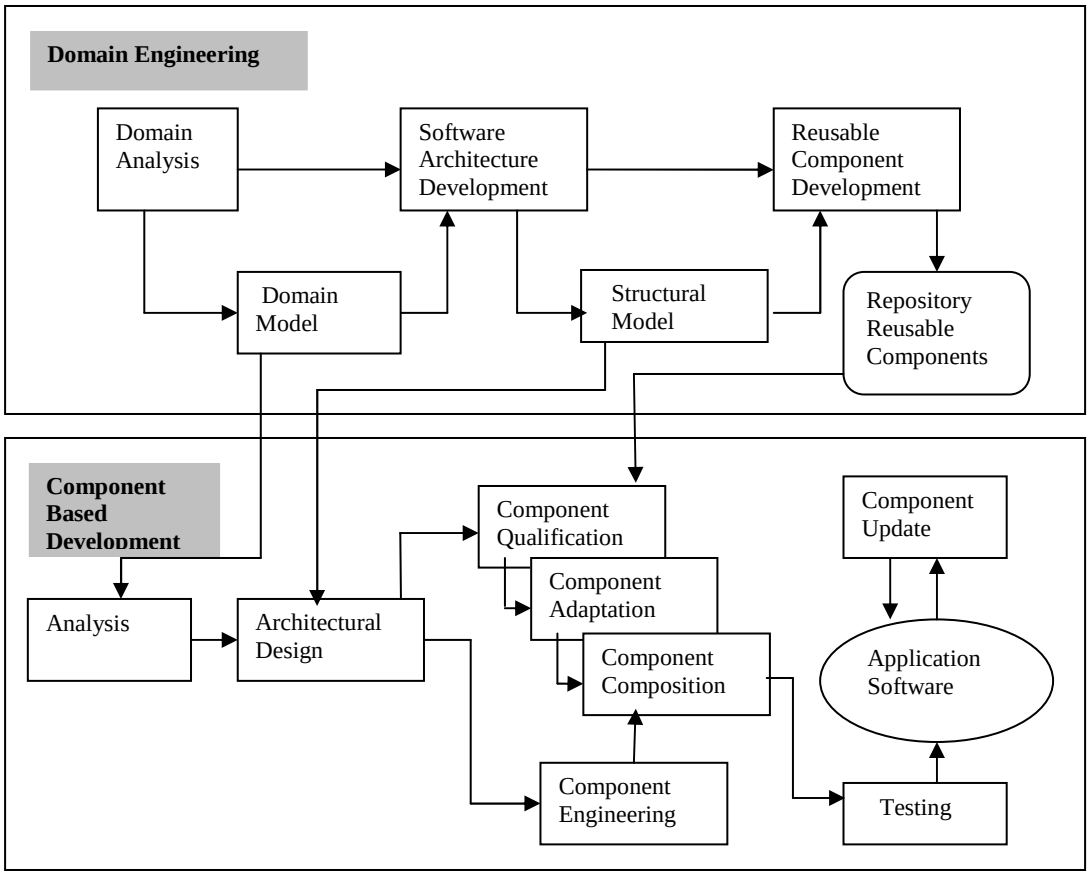


Figure 1.1: CBSE Process

1.6.1 Domain Engineering

The Domain Engineering, also called development "for reuse", deals with the components construction. Based on some problem domain requirements, the components are constructed for later reuse. A CBSE must support domain analysis, components design, implementation, packaging and publishing.

"Domain Engineering is about finding commonalities among systems to identify Components that can be applied to many systems, and to identify program families that are positioned to take fullest advantage of those components".

Development for Reuse: “addresses the accumulation of potentially reusable software objects in an organizational experience base... [This] involves mechanisms for:”

· Developing new objects or extracting objects from completed projects.

· Qualifying the objects.

· Recording the object for the experience base.

· Packaging (generalizing, tailoring, and formalizing) the objects.

· Providing feedback to improve process.

1.6.2 Component Based Development

The Component-Based Development, also called development “with reuse”, deals with the applications construction. These applications are composed by reusing existing components. If needed, new components can be developed, or even acquired from third party. An important issue in CBSE is that it is more desirable to reuse an existing component than to develop a new one, even if the requirements are not entirely attended. This reduces both cost and time to market, but requires a searching mechanism to access all the available components. Thus, to be effective, a CBSE must provide ways for finding, connecting and adapting existing components. It must also support the addition of components that are developed or acquired during this phase.

Development with Reuse: “assumes the existence of a sufficient number of potentially reusable software objects in an experience base...[This] involves mechanisms for:”

- Specifying reuse requirements.
- Retrieving candidate objects from the experience base that match requirements.
- Evaluating the selected objects for suitability and possibly selecting one or more.
- Modifying the primary object to correspond with the requirements.
- Integrating the object into the desired environment.

There are three stages in this process. These are

- Component Qualification
- Component Adaptation
- Component Composition.

(a) Component Qualification

Component Qualification examines reusable components. These are identified by characteristics in their interfaces, i.e. the services provided, and the means by which consumers access these services. This does not always provide the whole picture of whether a component will fit the requirements and the architectural style. This is a process of discovery by the Software Engineer. This ensures a candidate component will perform the function required, and whether it is compatible or adaptable to the architectural style of the system. The three important characteristics looked at are performance, reliability and usability.

(b) Component Adaptation

Component Adaptation is required because very rarely will components integrate immediately with the system. Depending on the component type (e.g. COTS or in-house) different strategies are used for adaptation (also known as *wrapping*). The most common approaches are:

White box wrapping: Here, the implementation of the component is directly modified in order to resolve any incompatibilities. This is, obviously, only possible if the source code is available for a component, which is extremely unlikely in the case of COTS.

Grey box wrapping: This relies on the component library providing a component extension language or API that enables conflicts to be removed or masked.

Black box wrapping: This is the most common case, where access to source code is not available, and the only way the component can be adapted is by pre / post - processing at the interface level.

It is the job of the software engineer to determine whether the effort required to wrap a component adequately is justified, or whether it would be "cheaper" (from a software engineering perspective) to engineer a custom component which removes these conflicts. Also, once a component has been adapted it s necessary to check for compatibility for integration and to test for any unexpected behavior which has emerged due to the modifications made.

(c) Component Composition

Component Composition integrated the components (whether they are qualified, adapted or engineered) into a working system. This accomplished by way of an infrastructure which is established to bind the components into an operational system. This infrastructure is usually a library of specialized components itself. It provides a model for the coordination of components and specific services that enable components to coordinate with one another and perform common tasks. Many of the features of a software system come from the interactions of its components. It is easy to see that the creation of features from component architecture can be thought of as careful component composition. Currently, there are two standard mechanisms to compose components: interconnection languages and standardized interfaces.

Interconnection Languages: Interconnection languages are special purpose description languages that describe how components are connected. At a minimum, the interconnection language describes the general structure of a system. Advanced interconnection languages describe the functionality of the system as well as communication protocols and connector properties. Interconnection languages have the benefit of abstractly specifying components in a separate modeling language. Although this improves the clarity of the model, it adds the burden of updating the system specification whenever a components definition changes or whenever components are composed differently. The most basic variety of an interconnection language is a module interconnection language (MIL).

***Standardized Interfaces*:** The method in which components are composed differentiates standardized interfaces from interconnection languages. In interconnection languages, connections between components are explicitly specified in a higher level language. Standardization allows a component to explicitly invoke a service defined in a standardized interface and then be composed with any component that realizes that interface. A standardized interface is developed for each realm. Finally, components are developed to implement the interface of a specific realm. Components that explicitly reference the standardized interface of other realms are said to be parameterized by those realms. Component composition is the process of replacing a realm parameter by a specific component from that realm.

1.7 DESIGN FOR REUSE

If specification matching yields components that fit the need of the current application, the designer can extract these components from a reuse library (repository) and use them in the design of new systems. If design components can not be found, the software engineer must apply conventional or object oriented method to create them. It is at this point-when the designer begins to create new components-that design for reuse (DFR) should be considered.

Number of issues that form a basis for design for reuse-

Standard data: The application domain should be investigated and standard global data structure should be identified. All design components can then be characterized to make use of these standard data structure.

Standards interface protocols: Three levels of interface protocols should be established: the nature of intra-modular interfaces, the design of external technical interfaces, and the human/machine interfaces.

Program templates: The structure model can serve as a template for the architectural design of a new program.

The development of a componentware is actually a process of developing software, and therefore, can also be divided into such sub-phases, namely, component specification, component design, component implementation, component testing, and component maintenance [18].

***Component Specification*:** These serve as contracts between customers and manufacturers of software systems. For complex systems or components, requirements analysis may be necessary. This involves steps like problem analysis, document analysis, data analysis, feasibility study, etc. Software components are typically less complex and do not always requires all these activities. Functional and nonfunctional requirements are among the most important parts of a specification. Other parts include user interfaces, error behavior, acceptance criteria, system environments, etc.

Functional requirements: Services that are expected by end-users.

Nonfunctional requirements: Constraints under which software has to operate.

Specifications should completely and consistently define the requirements on components. A component's specification not only serves as a contract between component developer and component user but also as a valuable source of information for evaluating a component's use value in a certain context.

Component Design: After component requirements have been analyzed and specified, a design has to be made. Component design is an iterative process and involves describing a component at different levels of abstraction. Design includes various activities:

Component or interface design defines component's interface in detail.

Data structure design defines data structures that are used for the implementation

Algorithms design defines the algorithmic decomposition of components.

Top-Down design is typical for components being built from scratch. Tasks are decomposed into subtasks until these can easily be formulated as algorithms.

Bottom-Up design proceeds in the opposite direction. Fundamental components are defined first and used to realize the next level of abstraction. Each level comprises what is called an abstract machine. Bottom-Up design is essential for the reuse of existing components.

Good design is crucial for the quality of a software component. There is no exact definition of what a good design is, but good designs are considered to have the following characteristics

Modularity: A component should be logically partitioned into subcomponents that perform specific functions.

Coupling: Loosely coupled components are as independent of other components as possible. For example, they do not have a shared state and do not interchange control information with other components.

Cohesion: Cohesive components represent single entities including all operations on these entities.

Understandability, adaptability: To make components understandable and adaptable, they should be loosely coupled and well documented.

Various categories of design methods exist, for example, function-oriented (like stepwise refinement), data-oriented (like Jackson Method) and object-oriented methods as proposed by Booch and Rumbaugh.

***Component Implementation*:** Implementation is the process of transforming a design into an executable form. For software systems this typically means coding in a certain programming language. According to definition, software components can have a variety of forms. They can be implemented in a programming language or be composed of components of any kind.

Ideally, the design of a component is independent of its implementation. In practice, however, this is often impossible. For example, we have to consider reusable components already during the design phase. Depending on what kind of components we consider, we make assumptions about the implementation of the component under development.

Component Testing: The purpose of testing is to ascertain whether a component satisfies its requirement by discovering as many errors as possible. Tests can be applied to different aspects of components. Various kinds of tests include the following:

Specification test: Specification tests check for the completeness, clarity, consistency and feasibility of a component specification. This could be done together with potential component users.

Component test: Component test reveal discrepancies between a component's specification and its implementation.

Integration test: Composing tested components can reveal new kinds of errors that stem from the interaction of components. Integration tests are applied to sub-systems or components being composed of lower-level components.

Acceptance test: The development of software products ends with an acceptance test where real operating conditions are used. This may not be possible for software components where potential reuse candidates might not even be known at the time.

Various testing methods and strategies include:

Static/dynamic testing: Static testing involves activities to find errors via static and semantic analyses. For dynamic tests components have to be executed or simulated.

Black-Box/White-Box testing: Black-Box tests involve input/output relationships of components. White-Box tests consider the inner structures of components as well.

Top-down/Bottom-Up testing: In top-down testing the main components are tested first by using stubs for components that are not yet available. In bottom-up testing basic components are tested first, followed by higher-level components that rely on lower-level components. With software component, software quality can be increased and testing efforts can be decreased.

***Component Maintenance*:** Component maintenance is the modification of a software component after its first delivery. Such modifications include error corrections, performance or other improvements, functionality extensions, or adaptations to changed environments. Component maintenance is by far more than just fixing bugs. Maintenance activities fall into the following categories:

Adaptive maintenance: To make a software component usable in a changed environment, e.g., adapting a component to a new version of an operating system or application framework.

Corrective maintenance: To overcome existing errors, i.e., diagnosing and correcting bugs.

Perfective maintenance: To improve performance, maintainability, or other quality attributes, e.g., enhancements demanded by users.

Preventive maintenance: To prevent future maintenance activities, e.g., redesigning, recording, and/or re-testing, sometimes complete re-engineering.

1.8 BENEFITS OF CBSE

The advent of component software is one of the most important new developments in the software industry since the introduction of high-level programming languages. Component software combines the advantages of custom software and standard software. It enables solutions that are better evolvable, are more readily maintainable, can be extended over time, and can be modernized incrementally.

1.8.1 Business Benefits of CBSE

Business benefits of component based software engineering are

Short Time to Market: Applications with components can be delivered synchronously with the business change cycle time

Adaptable: Componentized applications can be able to respond rapidly to change without forcing costly rewrites of the entire system.

Supporting Integration: There is often no time or justification to replace legacy applications. New functionality can be integrated with other packages, existing applications, and data sources.

Applicable: Componentized applications can align with the business. It is unacceptable to require the business to change to the application functionality.

Upgradeable: Improvements to an existing function can be possible without impact to other functions.

1.8.2 Technical Benefits of CBSE

In the view of component users, the Technical benefits can be summarized as followings:

Power users will find it second nature to assemble their own personalized applications using off-the-shelf components. They will use scripts to tie the parts together and customize their behavior.

Small developers will find that components reduce expenses and lower the barriers to entry in the software market. They can create individual components with the knowledge that they will integrated smoothly with existing software created by larger development shops - they do not have to reinvent all the functions around them. They can get fine grained integration instead of today's 'band-aid' integration. In addition, they get faster time to market because the bulk of an application is already there.

Large Developers, IS shops and System integrators will use component suites to create enterprise-wide client/server applications in record time. Typically, about 80% of the function they need will be available as off-the-shelf components. The remaining 20% is the value-added they provide. The client/server systems may be less complex to test because of the high reliability of the pre-tested components. The fact that many components are black-boxes reduces the overall complexity of the development process.

Desktop vendors will use components to assemble applications that target specific markets. Instead of selling monster suites at high prices, they will be able to provide their consumers with what they really

need. Even the *Pay-per-Use* can be achieved, when customers use the CORBA compatible products, which provide this CORBA Service. Increased customization and more malleable products will create new market segments. Consumers will not be at the mercy of long product release cycles to get new functions. They can buy add-on functions, in the form of components, when they need it.

1.9 SUMMARY

In this chapter concept related to component and component base software engineering (CBSE) is described. The complete CBSE process, and design for reuse has been discussed. Also various benefits of CBSE are explored. Various component retrieval techniques are considered in next chapter.

Chapter 2
Classification and Retrieval Techniques

A retrieval technique is concerned with the comparison of the representation of a query and the representations of documents for the purpose of identifying and or retrieving those documents that are relevant to that query. A retrieval technique is not directly concerned with the way of actually representing the documents and queries, but different ways of representation may imply using different retrieval techniques. In this sense representations and retrieval techniques are interrelated. In principle, information storage and retrieval is simple. Suppose there is a store of documents and a person (user of the store) formulates a question (request or query) to which the answer is a set of documents satisfying the information need expressed by his question. User can obtain the set by reading all the documents in the store, retaining the relevant documents and discarding all the others. In a sense, this constitutes 'perfect' retrieval. This solution is obviously impracticable. A user either does not have the time or does not wish to spend the time reading the entire document collection, apart from the fact that it may be physically impossible for the user to do so. Often also the term retrieval model is used. In the literature exists a lot of confusion about the difference between model and technique. In fact both terms are frequently used without, that it is really explained why. If we want to make some difference after all we could roughly say that a retrieval technique is mostly concerned with the matching of the document descriptors and the query representations. Whereas a retrieval model is going a little beyond this, also taking into account the way of representing the documents and the indexing process. Intellectually it is possible for a human to establish the relevance of a document to a query. For a computer to do this we need to construct a model within which relevance decisions can be quantified. It is interesting to note that most research in information retrieval can be shown to have been concerned with different aspects of such a model.

There exist a great number of retrieval models or techniques providing different approaches to the way of locating information.

2.1 THE RETRIEVAL PROCESS

In the retrieval process, the IR system extracts the documents or more generally, the pieces of information, which will presumably answer the information need formulated

by the user. This retrieval process is usually separated into a preliminary step of indexing the documents, followed by an operational step of retrieval. The retrieval process can be iterative, if the user provides some feedback on the retrieval results. Figure 2.1 shows a general view of the retrieval process.

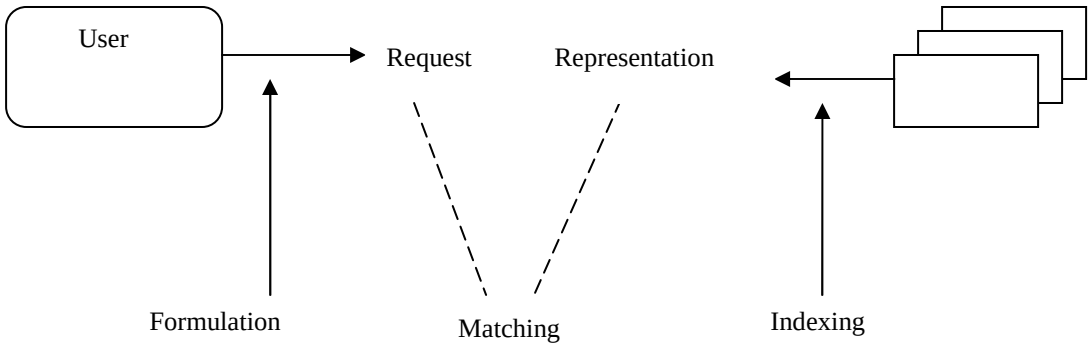


Figure 2.1: A General View of the Retrieval Process.

Information retrieval system has three basic issues, these are:

- The formulation of the query
- The indexing process
- The matching function

***Formulation*:** The problem of the formulation of a query is closely related to the issue of information need. Information need is distinguished into two types:-

- The extensional information need.
- The intensional information need.

An extensional information need is a clear and specific information need. The searcher knows precisely what he is looking for and where he can find it. For example the title of a book that has to be found in some catalogue. In this case the user is very well able to formulate his information need into a request since he knows very well what he wants. Consequently, his query is likely to be an exact representation of the information need. On the contrary, an intensional information need is much less clear and specific. The user does not know exactly what he is looking for; he usually only has a general idea about his information need. For example, he might be looking for information on a particular subject, but he does not know exactly what kind of information he requires. His information need is thus imprecise and probably incomplete.

***Indexing*:** Before an automatic system is able to retrieve any information, this information must first be stored in some way in the system. In general, it is unlikely that the full text of each document is stored in the system, especially if a large document collection is concerned. Therefore, each document is assigned a descriptor, which is a representation of its informative content within the system. The process by means of

which such a representation is generated is called indexing. Indexing can be done manually or automatically. Manual indexing is usually performed by trained experts. They scan the documents contents and accordingly assign descriptors on the basis of their knowledge and experience.

***Matching and Relevance*:** Once the user has formulated his information need into a request, this query is compared with the document representations. This comparison is called matching in Information Retrieval. The distinction is made between exact matches and partial matches. In case of an exact match the query representation and the document descriptor must be equal in order to retrieve the document. In case of a partial match, the document is retrieved when the representation of the document is similar enough to the representation of a query. Here we touch upon the matter of relevance. When a document is 'similar enough' to a query, it is 'relevant' with respect to that query. Relevance is a very important notion in Information Retrieval, since the main purpose of Information Retrieval is to retrieve as many relevant documents as possible in response to a user request. Since relevance is a subjective notion, it is very difficult to define what is relevance. Users often only have a general idea about what kind of information they request on a particular subject. Different users probably have different ideas about which documents are relevant to them and which documents are not. Often they are not able to formally define why they consider a particular document relevant. If you would ask them, they would probably reply "It looks interesting; there might be something in it". Furthermore, relevance is context dependent. The context of a document within a collection of documents is the subset of documents that are somehow related to that document, for example through bibliographical citations or because of a certain overlap in their informative contents, which is expressed in the usage of the same index terms in the different document representations.

2.2 MAIN APPROACHES FOR SOFTWARE

CLASSIFICATION AND RETRIEVAL

Various Software classification and retrieval techniques are available; some of them are discussed in this section. These are:

- Classification Schemes
    - Enumerative Scheme
    - Faceted Classification
- Formal Specifications
- Knowledge- Based Approach
- Hypertext
- Browsing
- Behavior Based Retrieval
- Automatic Indexing

- Soft Computing Techniques
    - Fuzzy Logic Based Retrieval
    - Neural Network Based Approach
    - Genetic Algorithm Based Retrieval
- Probabilistic Retrieval

2.2.1 Classification Schemes

A fundamental problem in software reuse is organizing collections of reusable components for effective search and retrieval. There are two main classification schemes- enumerative scheme and faceted classification.

2.2.1.1 Enumerative Scheme

Classification schemes such as Library of Congress and Dewey Decimal are Enumerative. In an Enumerative scheme all possible classes are pre- defined. The librarian selects the class that best fits the attributes of the new item by traversing the classification hierarchy.

In (entities) the Dewey Decimal System, for designs example, if we want to classify the programs title "Structured Systems Program- structures ruing," we look at the hierarchical systems structure and try to find the predefined class that best describes the title. A partial expansion of the Dewey hierarchy is shown in Figure 2.2.

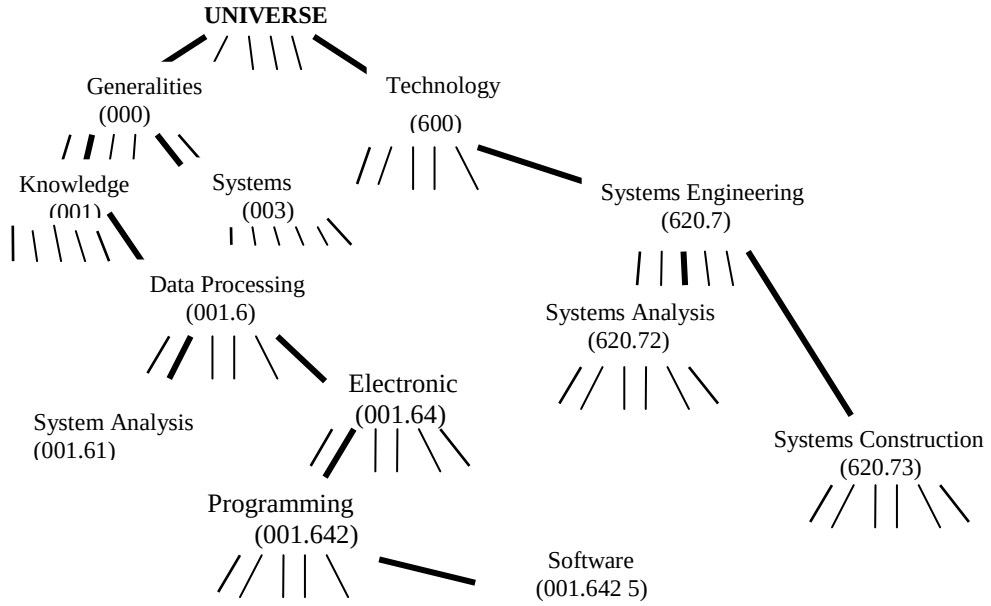


Figure 2.2: Partial Expansion of the Dewey Decimal Hierarchy

Structured Systems Programming could match any of the following classes: 001.61 (systems analysis), 001.642 5 (software), 003 (systems), 620.72 (systems analysis), or 620.73 (systems construction). Implicit in enumerative classification is the expertise of the librarian in both the classification scheme and the subject matter or domain of the title. Deciding which the most appropriate class is a difficult task. The title in this example is entered under 001.642 5 (software). Cross-references establish links among related classes to simplify the class selection and to facilitate searching, but they add a complex network of relationships that is difficult to represent and maintain.

2.2.1.2 Faceted Classification

Faceted classification offers certain features that not only improve search and retrieval, but support the potential reuse selection process and contribute to the development of a standard vocabulary for software attributes. Classification in a faceted scheme is straightforward. A faceted scheme may have several facets and each facet may have several terms. A hypothetical faceted scheme with two facets, entities and activities, is introduced here to illustrate the classification process [36].

Table 2.1 Faceted Schemes

{entities}	**{activities}**
designs	analysis
programs	design
structures	evaluation
systems	programming

To classify the title in our example, we select from each facet the term that best describes each of the concepts in the title. "Systems" is selected from the entities facet, and "programming," from the activities facet. The resulting synthesized class is systems/programming. Faceted classification tailors the class to a perfect fit, while in enumerative schemes the librarian must shop for the best fit from a collection of standard sizes.

Table 2.2: Simplified Faceted Scheme for UNIX Components

DOMAIN → UNIX tools			
{by action}	**{by object}**	**{by data structure}**	**{by system}**
get	file-names	buffer	line-editor
put	identifiers	tree	text-formatter
update	line-number	table	
append	character	file	
check	number	archive	
detect	expression		
locate	entry		
search	declaration		

evaluate	line
compare	pattern
build	

While the previous example deals with a hypothetical faceted scheme, the example in Table 2.2 is a faceted scheme developed for UNIX components. It consists of four facets: the function performed by the component, the objects manipulated by the function, the data structure where the function takes place, and the system to which the function belongs. The example shows the class of UNIX components that locate line numbers in a file and are part of a line editor.

Why Faceted Classification?

Current approaches to information retrieval are often classified along a scale whose end points are free-text analysis and controlled vocabulary. Free-text analysis, also referred to as uncontrolled vocabulary, consists in analyzing word frequencies in natural text. Relevant keywords are derived automatically by their statistical and positional properties, thus resulting in what is called automatic indexing, which is the assignment of preferred terms to represent or define an item. A fundamental assumption in this approach is the existence of a large "corpus" of text to justify the statistical analysis. This technique has proven relatively effective for text intensive documents such as books and journal articles.

On the other hand, controlled vocabulary approaches rely on a predefined set of keywords used as indexing terms. These keywords are derived and defined by experts and are designed to best describe or represent concepts relevant to the domain of discourse. Software products have certain characteristics that make controlled vocabulary a more attractive approach over free-text analysis:

- Source code is very low on free text,
- Keyword meanings are usually assigned by convention or by programmer preference,
- It is not obvious "what" components do and "how" they do it, and
- Human intervention is typical when describing component functionality.

In a controlled vocabulary, keyword-represented concepts are organized as a classification scheme. A classification scheme provides a network of predefined relationships, thus introducing some semantic information absent in free-text analysis.

A classification scheme for collections of reusable software components should meet the following criteria:

1. It must accommodate continually expanding collections, a characteristic of most software organizations.
2. It must support finding components that are similar, not just exact matches.
3. It must support finding functionally equivalent components across domains.
4. It must be very precise and have high descriptive power (both are necessary conditions for classifying and cataloging software).
5. It must be easy to maintain, that is, add, delete, and update the class structure and vocabulary without need to reclassify.
6. It must be easily usable by both the librarian and end user.

7. It must be amenable to automation.

Faceted classification meets most of these demands. Expansion and maintenance (1) is easily accomplished by adding and updating the faceted lists. Precision and descriptive power (4) is a consequence of the synthetic approach to classification. A consistent list of terms for each facet provides for a standard vocabulary that is common to both the librarian and the user (6), and its tabular format makes it easy to implement as a relational database (5,7). Therefore, a simplified faceted approach was selected for reusable software over the classical enumerative schemes. A conceptual graph structure was superimposed on each set of facet terms to measure class similarity (2, 3) and a thesaurus was added to each facet. Thesaurus enhances query formulation and classification, support concept clarification, and simplify maintenance (2, 5, 6).

2.2.2 Formal Specifications

Using formal specifications to represent software components facilitates the determination of reusability because they more precisely characterize the functionality of the software, and the well-defined syntax makes processing amenable to automation.

Formal specifications use mathematical notation to describe in a precise way the properties which an information system must have, without unduly constraining the way in which these properties are achieved. They describe what the system must do without saying how it is to be done. This abstraction makes formal specifications useful in the process of developing a computer system, because they allow questions about what the system does to be answered confidently, without the need to disentangle the information from a mass of detailed program code, or to speculate about the meaning phrase in an imprecisely worded prose description [19, 23].

A formal specification can serve as a single, reliable reference point for those who investigate the customer's needs, those who implement programs to satisfy those needs, those who test the results, and those who write instruction manuals for the system. Because it is independent of the program code, a formal specification of a system can be completed early in its development. Although it might need to be changed as the design team gains in understanding and the perceived needs of the customer evolve, it can be a valuable means of promoting a common understanding among all those concerned with the system.

In this approach, queries are formal requirement specifications (e.g., the specification of a signature) and the system retrieves relevant software from a library of formally specified components by invoking a theorem prover to determine if component specifications satisfy the requirements.

2.2.3 Knowledge-Based Approach

The text automatic classification method is based on the content analysis automatically to allocate the text into pre-determined catalogue. The methods of text automatic classification mainly use information retrieval techniques. Traditional information retrieval mainly retrieves relevant documents by using

keyword-based or statistic-based techniques. Generally, three famous models are used: vector space model, Boolean model and probability model. One central step in automatic text classification is to identify the major topics of the texts [43].

Knowledge-based software retrieval systems make some kind of lexical, syntactic and semantics analysis of natural language specifications of software components without pretending to completely understand the document. They are based on a knowledge base which stores semantics information about the application domain and about natural language itself. These systems are usually more powerful than traditional keyword retrieval systems. However, they usually require enormous human resources: Knowledge bases are created for each application domain and are usually populated manually. The principal process for the Knowledge –based text classification is illustrated as following:

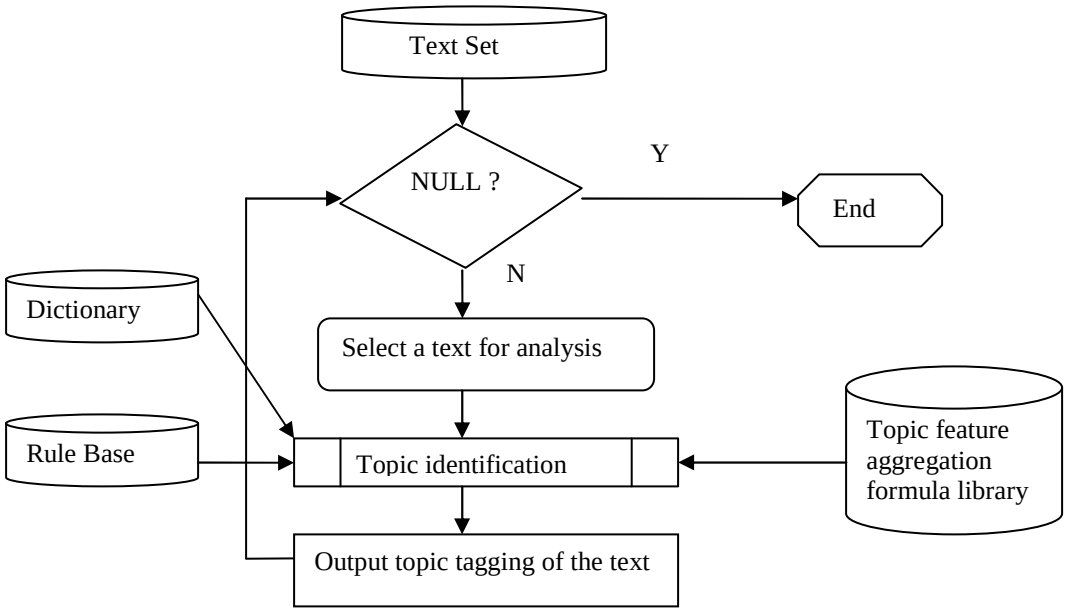


Figure 2.3: Knowledge-Based Text Classification

2.2.4 Hypertext

A hypertext consists of a set of interconnected units of textual information such a unit of information is called a node the connections between the nodes are called links The starting point of a link is called the anchor or the parent node. The ending point is called the destination or the child node usually links are one directional but it is also possible to define two dimensional links. Nodes can be anchors and destinations of more that one ink By means of this set of nodes and links a structure is imposed on the collection of data a browsing feature which has been classified as a retrieval technique earlier permits the user to wander through the data thus providing a natural way of accessing the collection of data.

Nodes can be of any nature, they could for example be titles of sections or chapters in a book or they could be bibliographical citations names of figures tables etc. Links are pointers from one node to another. They could be references to sections figures or table's citations etc. The actual information is stored in a database and the nodes of the hypertext correspond to pieces of the information. Only if a user finds a node interesting enough the actual information is retrieved. A hypertext is thus a network structure on top of the actual information in the document collection. Following the links in the network through browsing there may be several different ways to reach the same node i.e. there are several different paths to the same node from a certain starting point. Following one directional links usually means more refinement. At a certain point the user might find out he has gone into too much detail. It is then possible to back track along the path one has taken to that particular node. It is usually not possible to go back to a different anchor node than the one the user came from, if the node containing the requested information is found the search can be ended. The path from a starting point to the final destination node is called a search path. Figure 2.4 presents a simple example of a hypertext structure. The nodes are assumed to contain some kind of textual information the links represent some kind of relationship between these pieces of textual information. For the sake of this example it is not necessary to actually specify the nature of the nodes and links

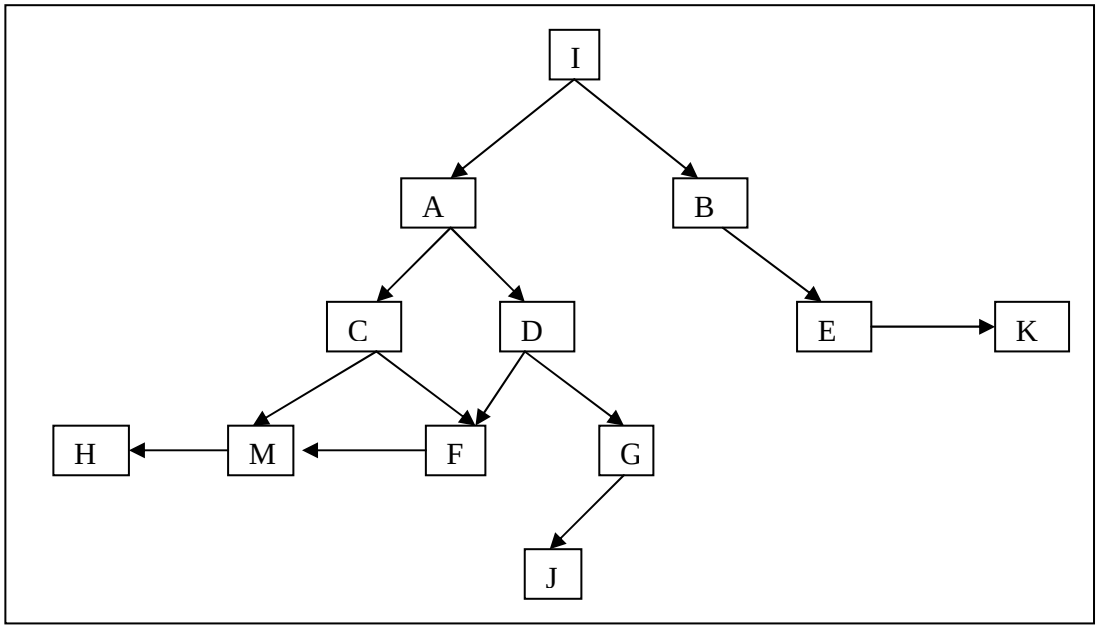


Figure 2.4: Hypertext Structure

Suppose the user takes node B as a starting point. He chooses to go to node E, but then finds out that is not what he was looking for. He back tracks to node B and chooses node D. Again there are two choices. Suppose he takes node F. If this would appear to be a wrong choice, he can back track to node D since this was the last node on his search path. It is not possible to go to node C, since this node has not been visited before and is therefore not on the search path. Having determined that F was a good choice, the user now decides to move to node M, which appears to contain the information he requested. Therefore, he ends his search at this point. The search path would in this case be B-E-B-D-F-M.

2.2.5 Browsing

Browsing is an example of a retrieval technique that requires a well structured document collection. Documents must be represented in the system as a network of inter connected nodes and a hypertext is a good way of defining such a structure. With the aid of the system, the user can now browse through this network to find the information he is looking for. The user does not have to formulate a query at first, to represent his information need and there is no such thing as a formal matching process. This can be a major advantage, especially when the user does not exactly know what he is looking for. In fact, the actual matching takes place in the uses mind while wandering through the network of nodes and links, he decides whether or not he finds the current node interesting. If so, he might want to retrieve the document belonging to the node and we could speak about a match. There are not formal rules for this kind of matching. The user may be changing his mind and decide otherwise or maybe he never really made up his mind about what he is looking for and is not able at all to find any matches[30].

As an example, suppose a user using some kind of drawing program wants to draw a star. He does not know how to do this or even if it is possible at all with the program. So he takes the manual and starts to browse through it to try and find an explanation on how to draw a star. In this case, the nodes could be the names of chapters or sections in the manual, while the links could be the cross-references between these chapters or sections. As a starting point, the user might look for the entry ‘star’ in the index. Reading the particular page referred to in the index, he finds out that it is only a general introduction. However, he finds a reference to a more specific section on the subject. In this way the user will continue to follow cross-references until he finds some description on how to draw a star, or until he finds out that it is not possible to draw stars with the program.

For very large collections of documents, browsing can be rather tedious and inefficient and in such cases querying might be a better idea. We could try to get the best of both and combine them.

A search based on browsing requires the user to be positioned in a structured database. The user moves within this database on the basis of information receive (Figure 2.5). At no stage is a subset of items ever isolated from the database for separate consideration; rather the user moves through the whole database focusing on the areas of immediate interest.

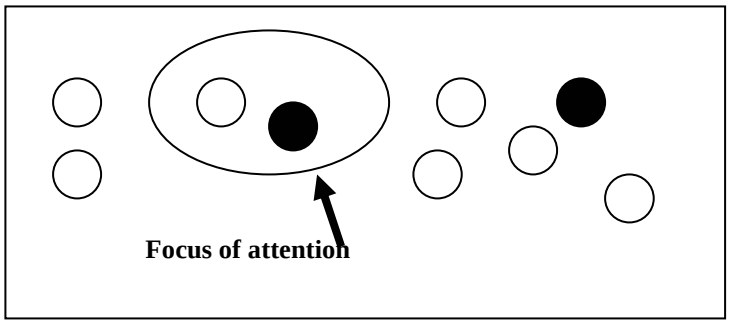


Figure 2.5: Browsing As Structure of Examination

The functional requirements for effective browsing systems are:

- The ability for users to position themselves in an area of interest of the database
- The ability for users to be able to recognize appropriate directions in which to further the search
- The ability to easily move quickly and efficiently through the database.

A key browsing requirement of a static database is for its structure to be understandable to the user and to have useful properties to help browsing. Browsing is fast because it is a pre-coordinate system, it is simple as there is only one basic operation, it is adaptable as it applies to any set of objects, it is versatile as different similarity conditions can be created, the user is in control and never needs to explicitly formulate a query and it gives a rich environment for creative browsing.

2.2.6 Behavioral Based Retrieval

Behavioral-based retrieval approaches are based on the notion of exploiting the executability of software components to classify them. Testing the component with different arguments calling his functions yields dynamic responses, which are collected. This collection is called the component behavior. An ordering on behaviors is then used to classify components and to search through the library of components. The programs used to produce the components behavior try to call a subset or all the functions of the component and recover the results. If the program calls functions that do not exist in the component, the components will ignore the call. In the behavioral-based retrieval approach, the engineer query is a program that calls some functions with specific arguments and this program will be plugged to all components of the library to test the components behavior. The components that respond to the searched behavior will be selected and presented to the engineer [29,34].

An Execution Model

In order to develop a theory of behavioural retrieval, an execution model is required to explain how components execute programs and generate output responses [40]. In execution model, a component is represented as a relation between programs and responses. This is becausand a response may be evoked by more than one program. Formally, a component *c* can be declared as:

$$\mathbf{c : prog \leftrightarrow Response}$$

A program **p є Prog** is modelled as a sequence of calls on the component's interface. A response is a sequence of values in correspondence with a program. Behavioural retrieval does not dictate a representation for effects; the representation of effects may or may not include descriptions of how the component state changed as a result of the call, or what information was shared with the environment.

Execution proceeds as follows: when sent a program a component will respond by executing each call in the program and producing a corresponding output response. This continues until all calls in the program have been executed or a call is rejected, in which case the execution ceases at that stage.

2.2.7 Automatic Indexing

Automatic indexing is the process of analyzing an item to extract the information to be permanently kept in an index. The system extracts lexical, syntactic and semantic information from the natural language description. These approaches automatically extract words or phrases (usually pairs of terms) from documents and from queries in natural language to build their internal representation. Automatic indexing is required for the effective retrieval of information. Indexing is essential to information retrieval because it provides entry points to a collection, without the user having to examine the whole collection. It tells the user where the information can be physically found and allows them to search a collection using keywords or phrases. Indexing has existed for as long as humans have been keeping written records. There are two ways that information can be indexed; manually or automatically. In manual indexing a human indexer compiles the index, while automatic is when the task is done by computer. The aim of automatic indexing is;

"...the capability for the system to automatically determine the index terms to be assigned to an item."

Systems that provide automatic indexing of software components can be classified in two basic groups: systems that work only at the lexical level and systems that include

Syntactic and Semantic analysis of software descriptions [26].

Lexical level: The system classifies software components according to attributes automatically extracted from their natural language documentation by using an indexing scheme based on the notions of lexical affinity and quantity of information. The retrieval system accepts queries in free style natural language and the same technique used to index software components is applied to queries. Even without attempting any understanding of the descriptions and without using any kind of syntactic and semantic knowledge, the system exhibits better precision than single keyword-based systems.

***Syntactic and Semantic analysis*:** These systems make some kind of syntactic and semantic analysis of natural language specifications without pretending to do a complete understanding of the descriptions. They are based upon a knowledge base which stores semantic information about the application domain and about natural language itself. These systems are more powerful than traditional keyword retrieval systems but they usually require enormous human resources because knowledge bases are created manually for each application domain.

2.2.8 Soft Computing Techniques

Soft Computing techniques are based on biological approaches to problem solving, where mathematics does not play as central a role as it does in engineering problem solving methods. Soft computing techniques are the innovative approaches for constructing computationally intelligent system that poses human like expertise within a specific domain, adapt themselves and learn to do better in changing environments. There are three main soft computing techniques: Fuzzy logic, Neural network, Genetic algorithms.

2.2.8.1 Fuzzy Logic Based Retrieval

This section presents the fuzzy classification and retrieval model based on features, i.e., term pairs describing the functionalities of reusable components. Fuzzy weight assignment to features is presented. The classification is supported by a Thesaurus where a fuzzy synonymia relationship is defined. We refer to object-oriented code and therefore will employ the terminology of object-orientation. Moreover, we denote as component a reusable unit of code (either a class or a class cluster) whose granularity has been chosen by the users [12, 16].

Fuzzy Behavioral Descriptors

The Component Descriptors of the SIB (CDs) have a standard object-oriented part exploiting the characteristics of the Telos SIB description language, and a fuzzy part of the following format:

List-of [feature: fuzzy weight]

Describing the behavior of a software component implies a certain degree of imprecision due to the multiplicity of characteristics usually exhibited even by simple components. A summarization is important for effectiveness in comprehending and possibly reusing code. A multiple descriptor of code components taking into account several behavioral characteristics is a viable approach to deal with this imprecision source.

Software behavior is described by two terms structures called features allowing for a richer semantics than usual single term description, while remaining simple enough to avoid the need for a formal grammar and semantics definition of a component description language. E.g., for object oriented code, method names are taken as the first term in the feature, and the first argument of the method as the second term. A software descriptor is a list of features describing the characteristics of a software component, that is, of a class, or set of classes, or library, chosen as a unit of reuse by the system maintainers. A sample extraction performed on classes belonging to the NIH library will be shown in next section. As far as design documentation is concerned, the procedure is described for ER schemas. However, this method can be readily extended to other design documentations, such as software modules or Abstract Data Types, which can be dealt with using a combination of this technique with the one illustrated above for object-oriented code. The purpose is to extract in a purely syntactical way a descriptor of ER schemas [13, 14].

The classification steps are the following:

1. The descriptor of an ER schema becomes a repository class named with the schema name
2. The following filter is applied for each entity E_i: the relevance degree of an entity E_i as the number of relationships in which E_i participates plus the number of its attributes. Then, only the entities E_r are kept whose degree exceeds (the integer part of) half of the maximum degree in the schema.
3. Features of the form (E_r, attr) and (E_r, E_j) are added to the descriptor, where the comma is the term separator in the feature, attr are all the attributes of the selected entities, and E_j varies among all entities connected to E_r. Duplications are avoided, i.e. the directions of the relationships are ignored.
4. For entities belonging to an is-a hierarchy, all related entities are taken, and features of the form (entity name, attribute) are extracted and added to the descriptor.
5. The descriptor is completed with features including the names (if any) of the relationships of the entities extracted in the previous steps, together with their attributes. The applied information filter aims at maintaining the "main" entities of a schema basing on is-a hierarchy and on the number of attributes and relationships. In order to handle classification and retrieval by means of descriptors, the features in the list are assigned a relevance measure expressed by a weight, a number between 0 and 1. The role of weights is clarified by the following considerations.

Usually, software components have a basic behavior (e.g., an array will be assigned the basic feature "random access") and a set of additional methods that are common to other components (e.g. "'print-element"). Furthermore, there can be other methods ("sort- self", where "self" is the standard auto-reference) supporting the basic behavior and characterizing only partially the component. For these reasons, the features are weighted in order to enhance the visibility of different behaviors. The mechanism of weights employs a frequency analysis technique of terms used in methods. The technique is taken from text analysis literature: it is an adaptation of the classical term weighting functions to the case of software which has only a skeletal text structure. The weights in a descriptor are viewed as a fuzzy set and fuzzy techniques are used for the retrieval.

2.2.8.2 Neural Network Based Retrieval

In Neural network models, information is represented as a network of weighted, interconnected nodes. In contrast to traditional information processing methods, neural network models are "self-processing" in that no external program operates on the network: the network literally processes itself, with "intelligent behavior" emerging from the local interactions that occur concurrently between the numerous network components. Neural network models in general are fundamentally different from traditional information processing models in at least two ways.

- First they are self-processing. Traditional information processing models typically make use of a passive data structure, which is always manipulated by an active external process/procedure. In contrast, the nodes and links in a neural network are active processing agents. There is typically no external active agent that operates on them. "Intelligent behavior" is a global property of neural network models.
- Second, neural network models exhibit global system behaviors derived from concurrent local interactions on their numerous components. The external process that manipulated the underlying data structures in traditional IR models typically has global access to the entire network/rule set, and processing is strongly and explicitly sequentialized [33].

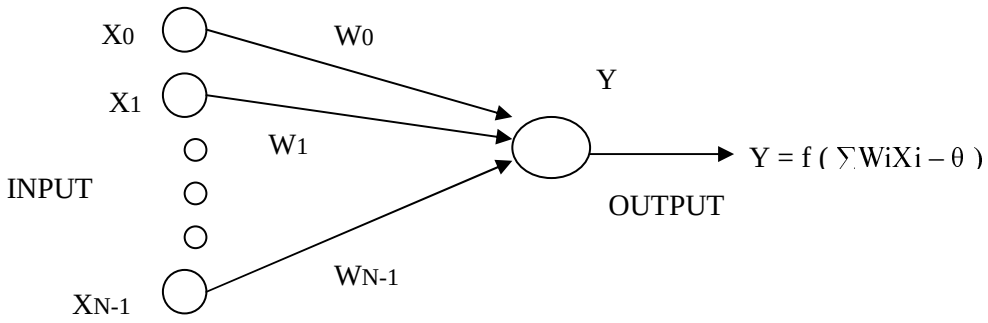


Figure 2.6: A Simple Neural Network.

A neural network has three components: a network, an activation rule, and a learning rule.

Network consists of a set of nodes (units) connected together via directed links. Each node in the network has a numeric activation level associated with it at time t. The overall pattern vector of activation represents the current state of the network at time t.

Activation rule is a local procedure that each node follows in updating its activation level in the context of input from neighboring nodes.

Learning rule is a local procedure that describes how the weights on connections should be altered as a function of time.

Similarly, Haykin (1999) thinks there are three basic elements of the neuronal model, which include:

A set of *synapses* or *connecting links*, each of which is characterized by a weight or strength of its own.
An *adder* for summing the input signals, weighed by the respective synapses of the neuron. The operations could constitute a linear combiner.
An *activation function* for limiting the amplitude of the output of a neuron. The activation function is also referred to as a squashing function in that it squashes (limits) the permissible amplitude range of the output signal to some finite value. Types of activation functions include: 1) threshold function; 2) Piecewise-linear function, and 3) sigmoid function. The sigmoid function, whose graph is s-shaped graph, is by far the most common form of activation function used in the construction of neural networks.
A Neural Network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns.

In a general sense, the use of neural networks offers the following useful properties and capabilities:
Nonlinearity: A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear. Nonlinearity is a highly important property, particularly if the underlying physical mechanism responsible for generation of the input signal is inherently nonlinear.
Input-Output Mapping*:* The network learns from the examples by constructing an input-output mapping for the problem at hand. Such an approach brings to mind the study of nonparametric statistical inference.
Adaptivity*:* Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment.
Evidential Response: In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to select, but also about the confidence in the decision made.
Contextual Information: Every neuron n the network is potentially affected by the global activity of all other neurons in the network.
Fault Tolerance: Its performance degrades gracefully under adverse operating conditions.

Neural networks computing, in particular, seems to fit well with conventional retrieval models such as the vector space model and the probabilistic model. Doszkocs et al. (1990) provided an excellent overview of the use of connectionist models in IR. A major portion of research in IR may be viewed within the framework of connectionist models. For example, to the extent that all connectionist models can be regarded as input-to-output classificatory systems, document clustering can be viewed as classification in the document*document space. Thesaurus construction can be viewed as laying out a coordinate system in the index*index space. Indexing itself can be viewed as mappings in the document*index space, and searching can be conceptualized as connections and activations in the index*document space.

Kinoshita and Palevsky (1987) predict that applying connectionist approaches to information retrieval will likely produce information systems that will be able to:

1) Recall memories despite failed individual memory units;

2) Modify stored information in response to new inputs from the user;

3) Retrieve "nearest neighbor" data when no exact data match exists;

4) Associatively recall information despite noise or missing pieces in the input,

5) Categorize information by their associative patterns.

2.2.8.3 Genetic Algorithm Based Retrieval

Genetic Algorithms (GAs) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of genetic algorithms is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem.

Genetic algorithms are good at taking larger, potentially huge, search spaces and navigating them looking for optimal combinations of things and solutions which we might not find in a life time.

Genetic algorithms were introduced as a computational analogy of adaptive systems. They are modeled loosely on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation-inducing operators such as mutation and crossover. A fitness function is used to evaluate individuals, and reproductive success varies with fitness [32].

All genetic algorithms follow some sequence of decisions that can be transformed into the following format, meant to mimic evolution by natural selection.

1) Generate an initial population

2) Select a “fit” subset of the organisms from the present population

3) Produce offspring by crossing different “fit” organisms

4) Allow mutations in the current population

5) Return to 2 until the termination criteria is achieved.

The paradigm of genetic algorithms described above is usually the one applied to solving most of the problems presented to GA’s. Though it might not find the best solution more often than not, it would come up with a partially optimal solution.

Nearly everyone can gain benefits from Genetic Algorithms, once he can encode solutions of a given problem to chromosomes in GA’s, and compare the relative performance (fitness) of solutions. An effective GA’s representation and meaningful fitness evaluation are the keys of the success in GA’s application. The appeal of GA’s comes from their simplicity and elegance as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. GA’s is useful and efficient when the search space is large, complex or poorly understood.

- Domain knowledge is scarce or expert knowledge is difficult to encode to narrow the search space.
- No mathematical analysis is available.
- Traditional search methods fail.

The advantage of the GA approach is the ease with which it can handle arbitrary kinds of constraints and objectives; all such things can be handled as weighted components of the fitness function, making it easy to adapt the GA scheduler to the particular requirements of a very wide range of possible overall objectives.

2.2.9 Probabilistic Retrieval

This model intends to retrieve documents according to their probability of relevance to the query. In the previously described retrieval models, a document is retrieved when it is in some way similar enough to a particular query. More specific, one can say that a document representation has a certain probability of relevance to the query. When the similarity between the query and the document representation increases, the probability of relevance also increases [28, 31].

In this model, document and queries are represented as n-dimensional binary vector (x_1, x_2 ,…., x_n) where x_i=1 or x_i=0 indicates the presence or absence respectively of terms in the representation. Given a certain query q for each document its probability of relevance is now expresses as: P_q (relevance | document)
It is assumed that the relevance of a document to a query is independent of the other documents in the collection. With this assumption, it is possible to introduce the probabilistic ranking principle. If the output of an Information Retrieval System is a ranking of the documents in decreasing order according to their probability of relevance to the submitted query, then the effectiveness of the Information Retrieval System will be optimal.

2.3 SUMMARY

In this chapter all types of information retrieval techniques are explored. Also the tasks associated with these classification and retrieval techniques are considered with their respective features. A genetic algorithm based best components retrieval model is proposed in next chapter.

Proposed Model

Information retrieval from components repository is a tedious work. The size of repository is often very large. Repository contains large number of components and its specification. Repository is a link between development for reuse, where the components are produced, and development with reuse, where components are reused.

For effectively reusing the components from the repository, the selection of proper retrieval technique is essential. Various linear search based retrieval techniques are available, which are used for information retrieval.

Optimizing the retrieved solutions is one of the key issues considered in proposed work. The most important goal of optimization is improvement, that how much improvement in the possible solution is achievable. The selection of best component or optimal solution from the retrieved components is much more difficult than the simple retrieval. Various soft computing techniques are available for optimization. Genetic algorithms are one of the soft computing techniques, which can be used to solve this problem. Genetic algorithms are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection.

Genetic algorithms are very different from most of the traditional optimization methods. Genetic algorithms work with a coding of variables. The advantage of working with a coding of variable space is that coding discretizes the search space even though the function may be continuous.

Genetic algorithms offer a robust non-linear search technique that is particularly suited to problems involving large numbers of variables. The genetic algorithm achieves the optimum solution by the random exchange of information between increasingly fit samples and the introduction of a probability of independent random change. Compared to other search methods, there is a need for a strategy which is global, efficient and robust over a broad spectrum of problems. The strength of genetic algorithms is derived from their ability to exploit in a highly efficient manner, information about a large number of individuals. Genetic algorithms take advantage of the old knowledge held in a parent population to generate new solutions with improved performance. An important characteristic of genetic algorithms is the fact that they are very effective when searching or optimizing spaces those are not smooth or continuous.

Genetic algorithms are iterative procedures that produce new populations at each step. A new population is created from an existing population by means of performance evaluation, selection procedures, recombination and survival. These processes repeat themselves until the population locates an optimum solution or some other stopping condition is reached, e.g. number of generation or time.

3.1 PROPOSED MODEL

The proposed system is an information retrieval system based on two step process. A component based repository is made, from which the best solution is to be found. The first step is getting the possible

solutions from the repository; this is done by keyword based retrieval. The user selects the keywords and on that basis all possible solutions are retrieved. The repository problem is that, search space is often complicated and one does not know where to look for the optimal solutions or where to start from and this is where genetic algorithm is useful. With the retrieval of possible solutions the first step is complete, these components now move into the next step for optimization. The next step is genetic algorithm based optimization. From these possible solutions, some components are randomly selected. On which the various genetic operators are applied. These operators are applied for number of generations or until the said condition is fulfilled. Once the genetic operations are complete, the components are ranked according to their fitness values. This gives the best components that can be reused.

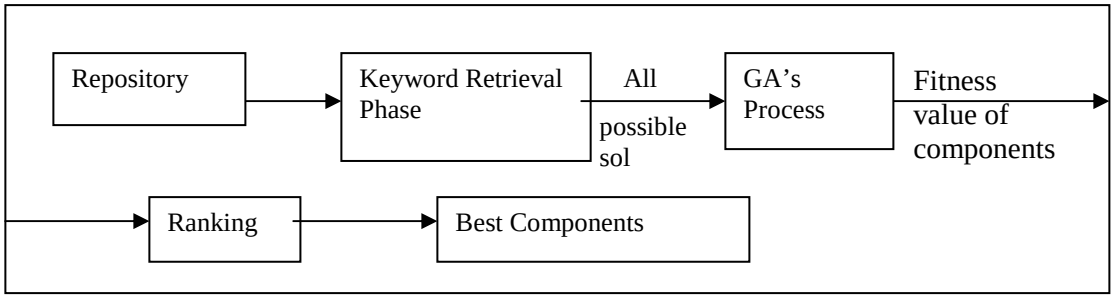
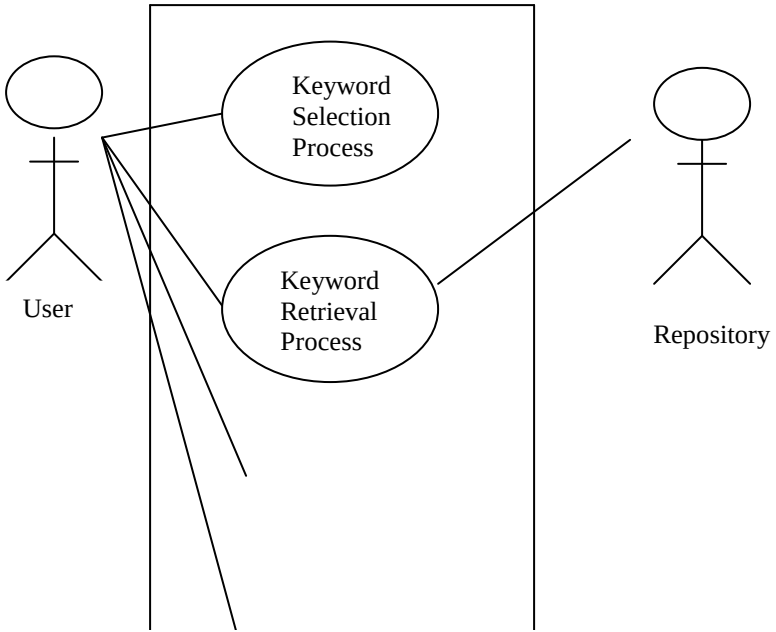


Figure 3.1: An Abstract Diagram of the Proposed Model

3.1.1 Modeling and Design

The UML is a language for visualizing, specifying, constructing, documenting the artifacts of a software intensive system. For the proposed system various UML diagrams are given.

- **Use Case Diagram**



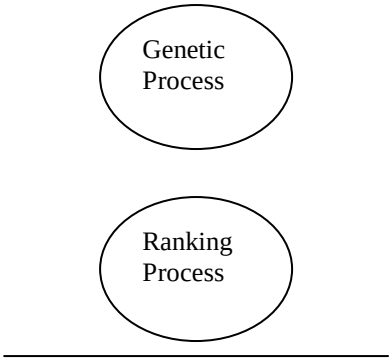


Figure 3.2: Use Case Diagram of the System

- **Class Diagram**

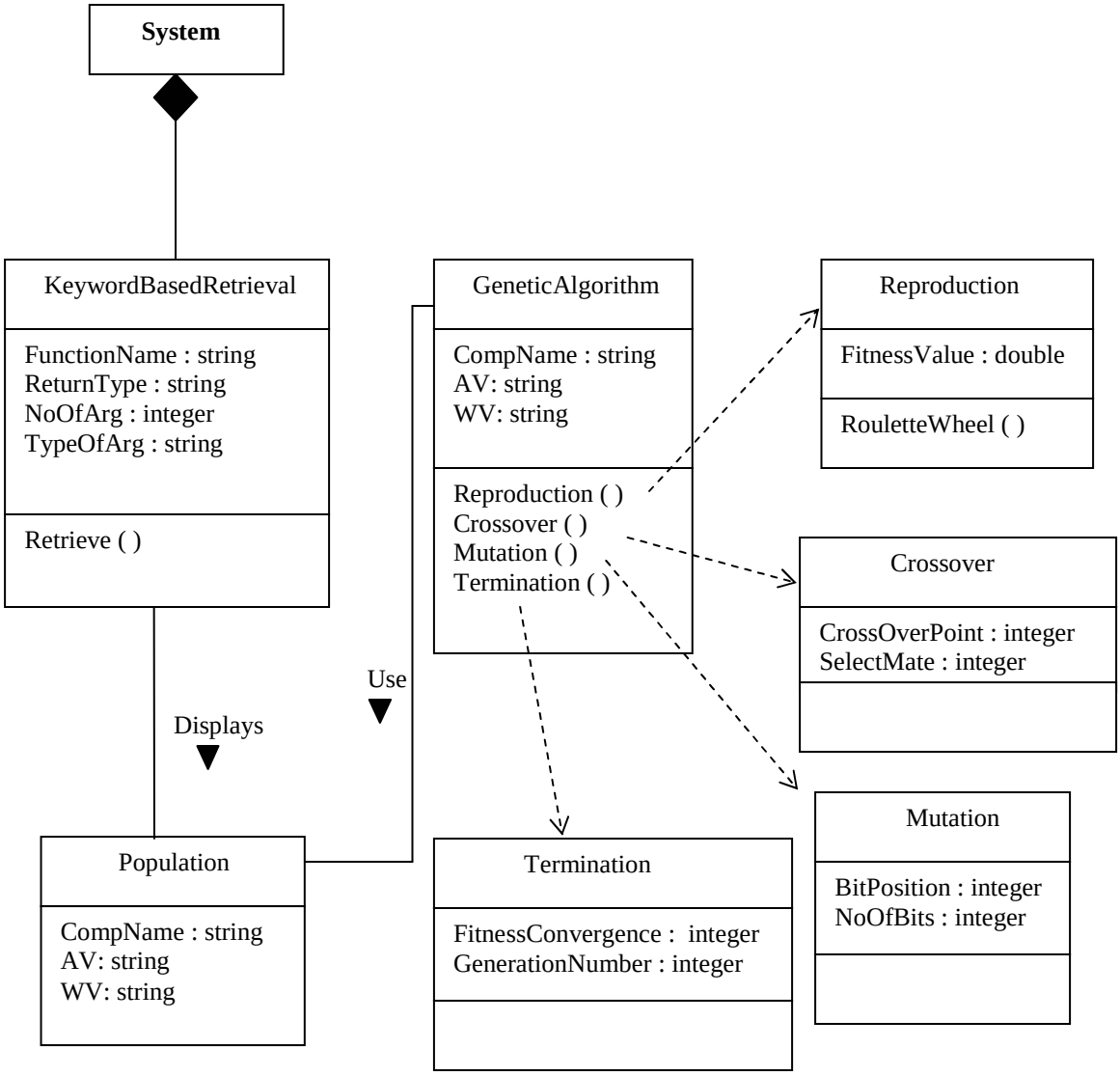


Figure 3.3: Class Diagram of the System

- **Sequence Diagram**

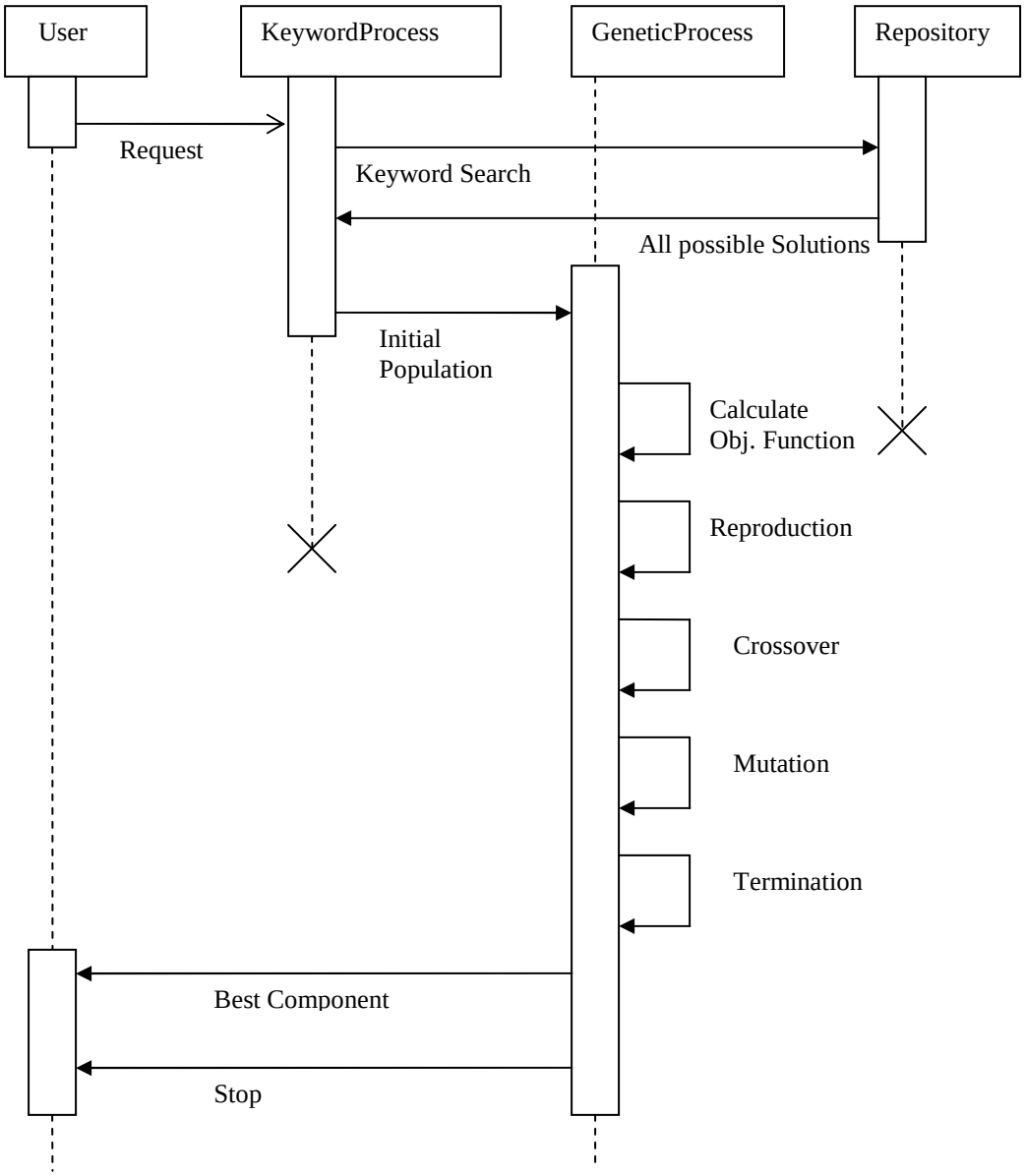


Figure 3.4: Sequence Diagram of the System

- **Activity Diagram**

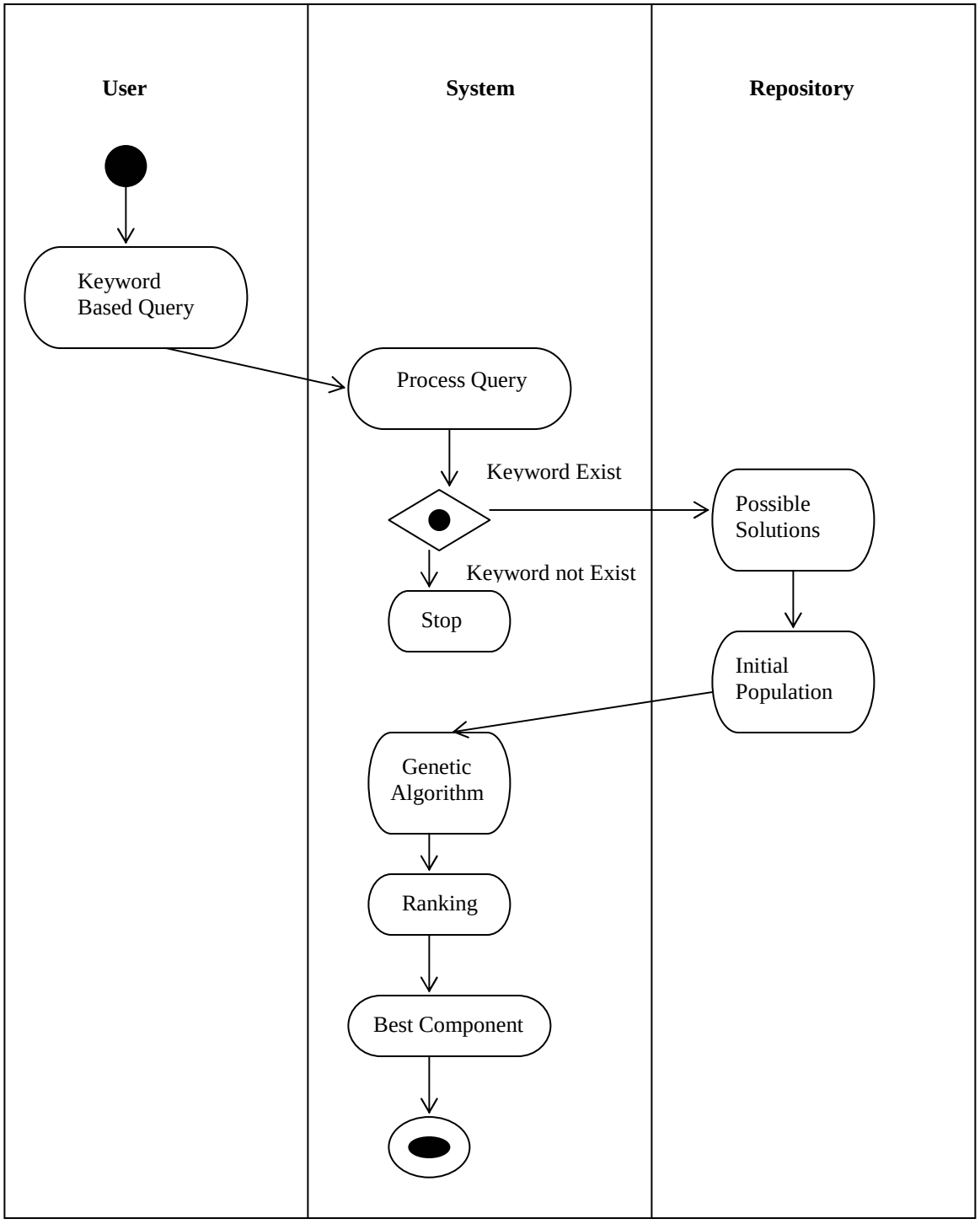


Figure 3.5: Activity Diagram of the System

- **Component Diagram:**

(a) Modeling a Physical Database

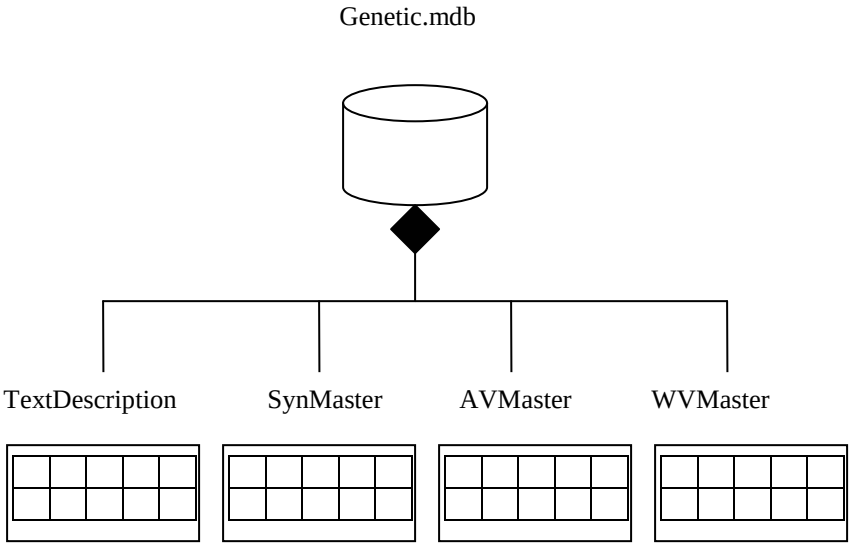


Figure 3.6: Component Diagram- Modeling a Physical Database

(b) Modeling Source Code

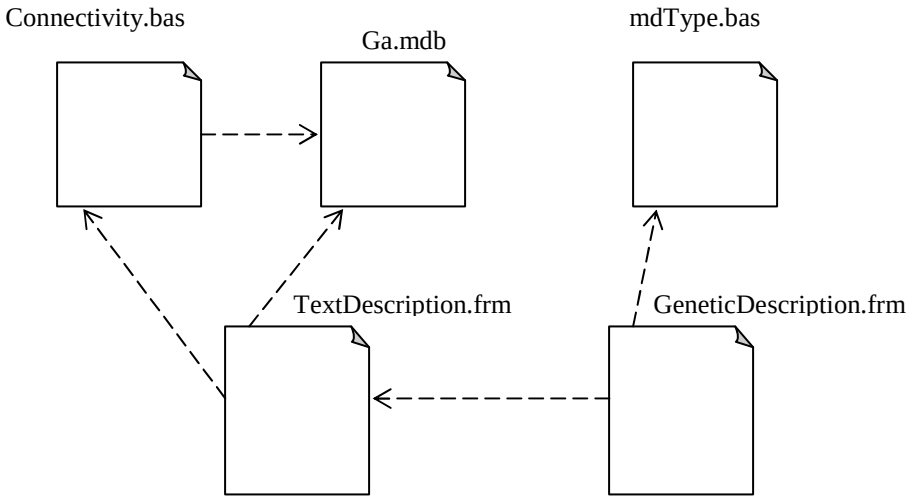


Figure 3.7: Component Diagram- Modeling Source Code

3.1.2 Component Description

Various components are stored in a database called as component repository. Our approach for component description is based on how software component is reused through the reuse model. Component retrieval becomes important as Component Based software development requires a large repository to supply

components. Repositories should provide highly sophisticated methods for storing and retrieving components. This is important because the size of a repository is very large. Proper specification of component in the repository results in easy retrieval of the component. System has been built with the following functionality.

-Descriptions of components are constructed using the classification framework.

-Users are able to search for components, based on the descriptions stored in the system.

The repository is the mechanism for defining, storing, accessing and managing all the information about a component. The repository allows software components and information about software component to be shared across software systems, life cycle phases. Efficient retrieval of component requires a proper classification of components. The classification is supported by a thesaurus where synonymia relationship is defined. All terms appearing in the thesaurus belong to the repository.

The component description is divided into three main parts.

- Component name
- Text description
- Genetic description

The component description used in the proposed model is given in Figure 3.8.

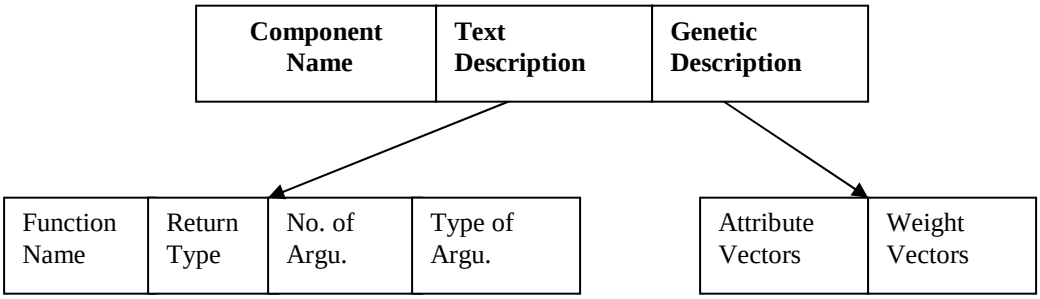


Figure 3.8: Component Description

Component name: It is a unique name given to a component in the repository. It has unique identity, without any repetition.

Text description: It focuses on the

-Function performed by the component and synonym corresponding to them.

-Return type used by the function,

-Number of argument used by the function,

-Type of arguments.

Genetic description: Genetic description of a component comprises attribute vectors and weight vectors. In the repository 8 attributes are assigned to a component, the attributes that are present, is assigned the value '1', whereas the attributes that are not present is assigned value '0'. The attributes vectors use binary coding. The weights corresponding to the attributes are also stored in this description. These weights give the importance of that attribute in the component. Value of weight vectors ranges from 0 to 1. In the Figure 3.9 Genetic description is shown, attribute vectors are given by 'A_i', where i =1 to 8 and 'Wi' represents the weight vectors. This type of weight vectors are widely used in the retrieval applications.

A1	A2	A3	A4	A5	A6	A7	A8

Attribute Vectors

W1	W2	W3	W4	W5	W6	W7	W8

Weight Vectors

Figure 3.9: Genetic Description

3.2 KEYWORD-BASED RETRIEVAL

Software reuse needs effective retrieval techniques to make development with reusable components more convenient than development from scratch. To this aim, components should be appropriately described. Localizing components in small software libraries can be simple. User can learn quickly where the available components are and can select them by name or browsing the library. Finding reusable components in large libraries is not as simple. Browsing the library can be very laborious and so mechanism allowing faster searching is needed to retrieve a component through the specification of its main features.

The proposed system uses keyword based retrieval for retrieving all relevant components. Keyword retrieval systems retrieve components through a set of keywords provided by a user employing a controlled vocabulary. The system based on a controlled vocabulary can exhibit reasonable retrieval effectiveness, depending on the quality of the vocabulary and on its flexibility to include new terms. Keywords systems are easier to create and to modify than classes systems. In fact they allow the addition of terms to cover new concepts without affecting existing terms.

Retrieval is assisted by the thesaurus containing unique terms and synonyms. A user often does not know the proper keywords. They may use a synonym or a related term or a more general or specialized term instead of the proper keyword. This increases the effectiveness of the keyword based retrieval in the proposed system. For keyword based retrieval, user has to select the keywords according to requirement from the interface. Figure 3.10 shows the keyword based interface. Then the system will match those keywords with the terms in the repository. The matched results are retrieved and are shown.

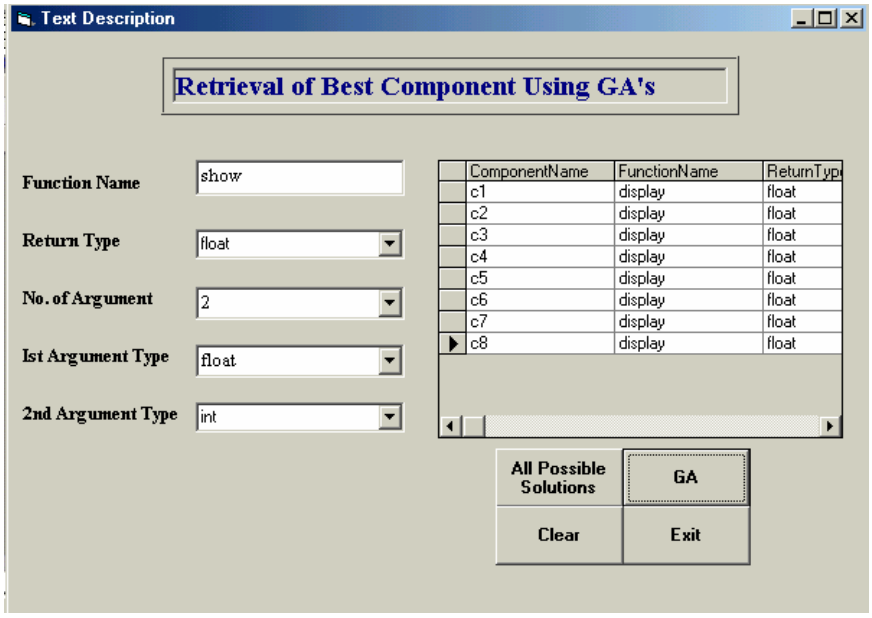


Figure 3.10: Keyword Based Interface

3.3 OPTIMIZATION USING GENETIC ALGORITHMS

Genetic algorithms are non-deterministic search algorithms based on the mechanics of natural selection and natural genetics in a biological system. Genetic algorithms have been used in modeling evolving systems or combinatorial optimization problems. Genetic algorithms are robust in many application areas and search a huge problem space while exploiting historical information to speculate on new search points with expected improvement of performance. This general strategy belongs to the class of probabilistic algorithm that use random choices and behave differently even when applied repeatedly on the same data. In this proposed system, the aim of the genetic algorithms is to optimize an information retrieval system.

The genetic algorithms attempts to find a best solution or optimal solution to the problem by genetically breeding the population of individuals. The genetic algorithm transforms a population of individual objects, each with an associated fitness value, into a new generation of the population using the Darwinian principle of reproduction and survival of the fittest and naturally occurring genetic operations such as crossover and mutation. Each individual in the population represents a possible solution to a given problem.

The keyword based retrieval gives all possible reusable components. The genetic description corresponding to these components are to be used for optimizing the search. In order to run Genetic Algorithms, an initial population is generated consisting of chromosomes and these chromosomes are evaluated using the objective function designed. In this two chromosomes are selected randomly, crossover and mutated them and replace a low quality chromosome with a new one of high quality. As these processes have been repeated, the population consists of high quality chromosomes.

Genetic algorithms approach comprises [42]:

- Initial Population

- Objective Function
- Genetic Operations
  - Selection and Reproduction
  - Crossover
  - Mutation
- Termination Criteria

Figure 3.11, shows the best component after applying genetic operators.

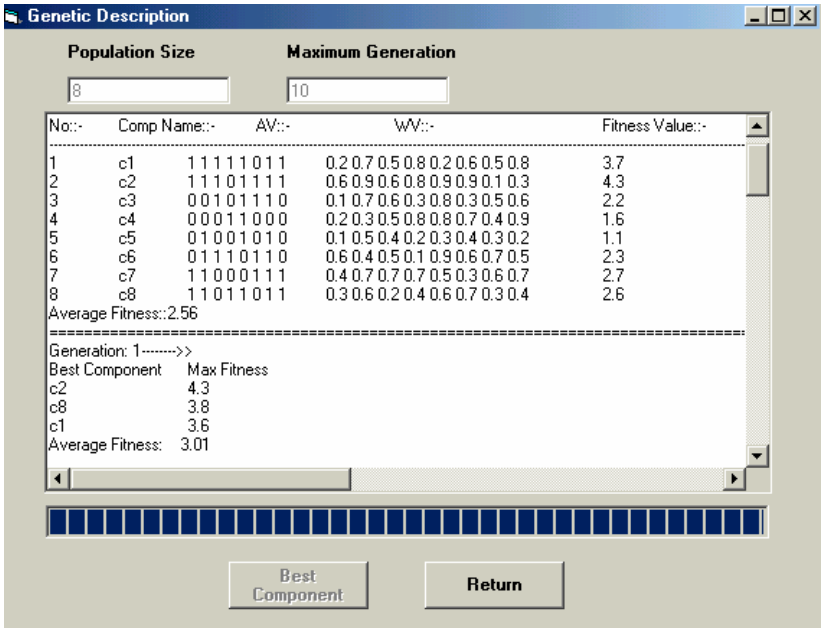


Figure 3.11: Optimization Using Genetic Algorithms

3.3.1 Initial Population

Genetic operators are applied on the initial population. A population is a set of chromosomes, in the context of Genetic algorithms; the chromosomes provide competing hypotheses on how to evaluate a document according to the information needs. Initial population is generated randomly. This one includes a genetic pool representing a group of possible solutions. Each chromosome in the population is of 8 genes or attributes. Each gene has weights assigned to them.

3.3.2 Objective Function

After initializing the population, Objective function of a chromosome determines how fit a chromosome is for each problem and which chromosomes survive in the next generation. Genetic algorithms favor chromosome which is of high quality. Objective Function in genetic algorithms is the function that is optimized using the genetic process. Choosing an appropriate objective function is very important. For each individual in the population, Fitness Value is evaluated as

$$\sum_{j=1}^{8} AjWj$$

'A_j' denote attribute vector W_j denote weight vector corresponding to that attribute vector, where 'j' denote total number of attribute in each chromosomes. It has fixed value i.e. 8. To get the best component the value of objective function must be maximized. Objective function of initial population is calculated, for primarily check. Generally, the objective function values increases after each generation.

3.3.3 Genetic Operations

In this step, genetic operators are applied to the individuals in the previous generation to generate the next generation of individuals. It involves these stages.

Selection and Reproduction: Genetic algorithm reproduction operator combine highly fit chromosomes to produce a more fit individual. Both the selection of parent chromosomes and the steps within the reproduction operator are randomized. Chromosomes are selected on the bases of the objective functions. The greater the objective function of a chromosome, the more likely the chromosomes will be selected for reproduction. The roulette wheel reproduction process is used to select individuals for reproduction. Roulette wheel selection is less noisy and is known as stochastic remainder selection. Each selected parent chromosome participates in the reproduction operation. All individuals in the previous generation are made available for reproduction in the next generation.

Crossover: A Single-site crossover is followed. In this a cross-site is selected randomly along the length of the mated strings and bits next to the cross-sites are exchanged [9].

Cross over is a recombination operator, which proceeds in three steps-

-Reproduction operator selects at random a pair of two individual strings for mating.

-Cross-site is selected at random along the string length.

-The position values are swapped between two strings following the cross-site.

Single-Site Cross Over

In a single-site cross over, a cross-site is selected randomly along the length of the mated strings and bits next to the cross-sites are exchanged as shown in Figure 3.12. If an appropriate site is chosen, better children can be obtained by combining good substances of parents. Since the knowledge of the appropriate site is not known and it is selected randomly, this random selection of cross-sites may produce enhanced children if the selected site is appropriate. If not, it may severely hamper the string quality.

Parent 1	p1	p2	p3	p4	p5	p6	***p7***	***p8***
Parent 2								

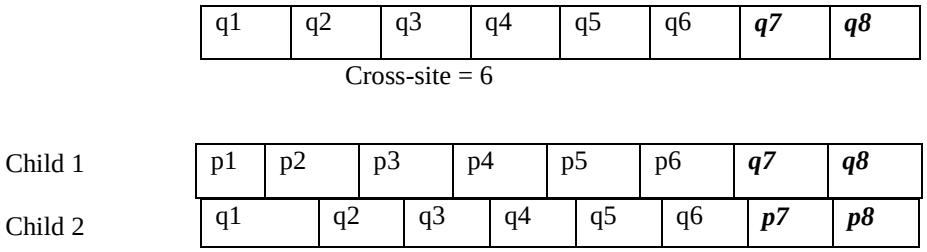


Figure 3.12: Single-Site Crossover

Mutation: In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In genetic algorithms mutation is randomly applied with low probability, typically in the range 0.001 and 0.01, and modifies elements in the chromosomes. Usually considered as a background operator, the role of mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero and acting as a safety net to recover good genetic material that may be lost through the action of selection and crossover. Mutation of a bit involves flipping it changing 0 to 1 and vice versa with small mutation probability P_m. The bit positions and their values are selected randomly for mutation.

Mutation Probability is the probability of mutation which is used to calculate number of bits to be muted. The mutation operator preserves the diversity among the population which is also very important for the search. Mutation causes movement in the search space and restores lost information to the population. Mutation probability used in the system is 0.02. The total number of the bits to be changed depends upon three parameter, mutation probability, population size and length of individuals in the population.

*Number of bits to be changed = Mutate Probability * Pop size * String length*

3.3.4 Termination Criteria

In genetic algorithms, the termination criteria for stopping the process is required. Genetic Processes has large number of iteration. Termination Criteria is important and vital step. This gives information about when to stop the process. Because the GA is a stochastic search method, it is difficult to formally specify convergence criteria. As the fitness of a population may remain static for a number of generations before a superior individual is found, the application of conventional termination criteria becomes problematic. A common practice is to terminate the Genetic algorithms, after a prespecified number of generations and then test the quality of the best members of the population against the problem definition.

Two Termination Criteria used in the proposed system are:

Fitness Convergence: A termination method that stops the evolution when the fitness is deemed as converged. Fitness is deemed as converged when the difference between average fitness across the current

population and previous population is less than the value specified. We have used 0.01 for termination criteria. Average fitness is calculated as,

$$\sum_{i=1}^{popsize} \sum_{j=1}^{8} AjWj \Big/ popsize$$

Where A_j denote attribute vector, W_j is weight vector corresponding to attribute vector and *popsize* is the population size.

$\text{AvgFitness}_{\text{current population}} - \text{AvgFitness}_{\text{previous population}} <= |0.01|$

When the said condition is satisfied the process will step.

Generation Number: In certain cases, above termination criteria is not achieved after large number of iteration, in that case another termination criterion is used i.e. Generation Number. Generation Number is a termination method that stops the evolution when the specified maximum numbers of evolution have been run.

Chapter 4

Conclusions and Future Scope

Effective retrieval of information from a repository is difficult and time consuming. Software reuse is based on effective information retrieval. In the absence of a proper retrieval mechanism the importance of the software reuse reduces drastically. The ultimate goal is to possess a manageable and predictable component retrieval technique. To enhance the information retrieval from a repository, a hybrid approach is proposed which consists of keyword based retrieval and genetic algorithms. Keyword based retrieval gives all the possible reusable components. Keywords systems are easier to create and to modify than classes systems and they allow the addition of terms to cover new concepts without affecting existing terms. To get optimal solution or best component from these components is the aim of genetic algorithm. Each component has attribute vectors and weight vectors assigned to them. After applying various genetic operators on weight vectors, fitness value of the component is evaluated that will identify the best component. A fitness

function is used for evaluating the relative merit of a component. Genetic operators are employed in an iterative process until a solution is found or a termination condition is met.

4.1 CASE STUDY AND EXPERIMENTAL RESULTS

In the proposed system, user gives query for component retrieval from the repository. The user selects the keywords in the keyword based interface. The keyword based retrieval will match those keywords with the repository keywords. After matching, all possible solutions will be displayed. On these possible solutions of reusable software components genetic algorithms approach will be implemented for optimizing the result. The repository consists of large number of components. In the given case study, 8 possible solutions are retrieved. With this retrieval of components the first step of the proposed system is terminated. All possible solutions are shown in Table 4.1. Possible solutions contain information like: Component name, Function name, return type, type of argument etc.

Table 4.1: All relevant components after keyword retrieval

	Comp1	Comp2	Comp3	Comp4	Comp5	Comp6	Comp7	Comp8
Comp Name	C1	C2	C3	C4	C5	C6	C7	C8
Function name	Display	Display	Display	Display	Display	Display	Display	Display
Return Type	Float	Float	Float	Float	Float	Float	Float	Float
No of Arg.	2	2	2	2	2	2	2	2
Type 1	Int	Float	Int	Int	Float	Float	Float	Int
Type 2	Float	Int	Float	Float	Int	Int	Int	float

Once the possible solutions are retrieved, the functioning of genetic algorithms starts for applying genetic algorithm operations; a randomly selected initial population is required, because genetic algorithms are based on probabilistic model. On this initial population all genetic operations will be applied. From the 8 possible solutions, 6 solutions are randomly selected; this becomes the initial population for the genetic process. The initial population selected randomly is shown in Table 4.2.

Table 4.2: Randomly selected initial population

C1
C2
C3
C5
C6
C8

Each component has certain attributes. There are total 8 attributes in the repository corresponding to each component. And each attribute has corresponding weights. Genetic operators are applied on the weight vectors. Table 4.3 and 4.4 shows attribute vectors and weight vectors corresponding to the components of initial population.

Table 4.3: Attributes of each component

Attributes / Component Name	A1	A2	A3	A4	A5	A6	A7	A8
C1	1	0	1	0	0	1	0	1
C2	0	1	0	0	0	0	1	0
C3	1	1	1	1	0	0	1	0
C5	1	0	0	0	1	1	0	1
C6	1	1	0	1	1	0	1	0
C8	1	0	0	1	0	1	1	0

Table 4.4: Weights assigned to each attribute in a component.

weights / Component Name	W1	W2	W3	W4	W5	W6	W7	W8
C1	0.2	0.6	0.5	0.8	0.5	0.6	0.4	0.5
C2	0.3	0.6	0.4	0.4	0.6	0.3	0.3	0.4
C3	0.8	0.1	0.5	0.5	0.8	0.6	0.8	0.5
C5	0.3	0.7	0.4	0.4	0.6	0.2	0.3	0.4

C6	0.5	0.1	0.5	0.3	0.8	0.5	0.7	0.5
C8	0.3	0.1	0.4	0.4	0.2	0.2	0.2	0.4

Fitness value of the each component is calculated using this formula.

$$\sum_{j=1}^{8} AjWj$$

Fitness values of components of initial population are shown in Table 4.5.

Table 4.5: Fitness value of each component

Component Name	Fitness Value
C1	1.8
C2	0.9
C3	2.7
C5	1.5
C6	2.4
C8	1.1

Average fitness is required , so that termination criteria can be obtained..

$$\sum_{i=1}^{popsize} \sum_{j=1}^{8} AjWj \Big/ popsize$$

where popsize = 6.

Average Fitness of initial population = (1.8 + 0.9 + 2.7 + 1.5 + 2.4 + 1.1) / 6 = 1.73

After calculating the average fitness of initial population, various genetic operators are applied to get the new generation. Various genetic operators used are

- Selection and Reproduction
- Crossover
- Mutation

New population generated after applying three operators, is shown in Table 4.6.

Table 4.6: New population after 1st generation

Comp Name	A1 A2 A3 A4 A5 A6 A7 A8	W1 W2 W3 W4 W5 W6 W7 W8	Fitness Values

C1	1	0	1	0	0	1	0	1	0.2	0.6	0.5	0.5	0.8	0.6	0.9	0.5	1.8
C2	0	1	0	0	0	0	1	0	0.2	0.6	0.5	0.8	0.5	0.6	0.4	0.5	1.0
C3	1	1	1	1	0	0	1	0	0.8	0.1	0.5	0.5	0.8	0.6	0.8	0.5	2.7
C5	1	0	0	0	1	1	0	1	0.8	0.1	0.5	0.5	0.8	0.6	0.8	0.5	2.7
C6	1	1	0	1	1	0	1	0	0.8	0.1	0.5	0.8	0.5	0.6	0.4	0.5	2.6
C8	1	0	0	1	0	1	1	0	0.8	0.1	0.5	0.5	0.8	0.6	0.8	0.5	2.7

Average Fitness after Generation 1 = 2.25

Two termination criteria's are used for stopping the process. These are:

1. $\text{AvgFitness}_{\text{current population}} - \text{AvgFitness}_{\text{previous population}} <= |0.01|$
2. Stop the genetic process after 100 generation, if first condition is not satisfied.

$\text{AvgFitness}_{\text{First population}} - \text{AvgFitness}_{\text{initial population}} = 0.52$

Average fitness difference between the first population and the initial population is more than the specified value in the termination criteria. So, the genetic process will continue

Table 4.7: New population after 27th and 28th generation

CompName	A1	A2	A3	A4	A5	A6	A7	A8	W1	W2	W3	W4	W5	W6	W7	W8	Fitness Values
C1	1	0	1	0	0	1	0	1	0.9	0.6	0.5	0.5	0.8	0.6	0.9	0.5	2.5
C2	0	1	0	0	0	0	1	0	0.9	0.6	0.5	0.5	0.8	0.6	0.9	0.5	1.5
C3	1	1	1	1	0	0	1	0	0.8	0.1	0.5	0.5	0.8	0.6	0.9	0.5	2.8
C5	1	0	0	0	1	1	0	1	0.8	0.1	0.5	0.5	0.8	0.6	0.9	0.5	2.7
C6	1	1	0	1	1	0	1	0	0.8	0.1	0.5	0.5	0.8	0.6	0.9	0.5	3.1
C8	1	0	0	1	0	1	1	0	0.8	0.1	0.5	0.5	0.8	0.6	0.9	0.5	2.8

Average Fitness after 27th and 28th generation:- 2.57

The genetic process stops after 28 generations, because the first termination condition is achieved.

$\text{AvgFitness}_{\text{after 28th generation}} - \text{AvgFitness}_{\text{after 27th generation}} = 0$

The first termination criterion is satisfied. Hence, the genetic process is stopped.

Best Components retrieved are ranked according to their fitness values. The best three components obtained are shown in Table 4.8.

Table 4.8: Best Components

Best Components	Max Fitness
C6	3.1
C3	2.8
C8	2.8

4.2 CONCLUSIONS

- Possible candidate components are selected using keyword search, Then genetic algorithms are applied to search the best components, so search does not start from blind rather search space is narrowed down first to reduce number of comparisons in the proposed technique.

- Appropriateness of the selected components will increase, as genetic search is based on attribute vector and weight vector.

- Genetic Algorithms are best suited for large search space; proposed technique has been tested with 5000 components, which can grow to any size.

4.3 FUTURE SCOPE

Potential avenues for future investigation include:

- Keyword based retrieval can be combined with another information retrieval technique to improve retrieval performance for candidate components.

- Genetic algorithms can be combined with other soft computing technique to get more optimized result.

- Text description of the components can be enhanced, for better results in the future.

Bibliography

1) Amit Singhal, “Modern Information Retrieval: A Brief Overview”, IEEE Computer Society Technical Committee on Data Engineering, (2001).
2) Axel Anders Kvale, “Empirical Study of Component Based Software Engineering With Aspect Oriented Programming”, (2004).
3) Bruce W. Weide, William F. Ogden, Stuart H. Zweben, “Reusable Software Components”, (1991).
4) Byung-Jeong Lee, Byung-Ro Moon, Chi-Su Wu, “Optimization of Multi-Way Clustering and Retrieval Using Genetic Algorithms in Reusable Class Library”.
5) C.J.Van Rijsenbergen, “Information Retrieval, 2nd Edition”, London: Butterworths, (1979).
6) Carma Mcclure,“The Three R’s of Software Automation- Re-Engineering, Repository, Reusability”, Prentice Hall, New jersey, (1992).
7) D.E.Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning”, Reading, MA: Addison-Wesley, (1989).
8) D.Vijay Rao, V.V.S.Sarma, “A Rough-Fuzzy Approach for Retrieval of Candidate Components for Software Reuse”, Pattern Recognition Letters 24, pp 875-886, (2003).
9) Dana Vrajitoru, “Crossover Improvement for the Genetic Algorithm in Information Retrieval, Information Processing & Management”, Vol. 34, No. 4, pp 405-415, (1998).
10) Darrell Whitley, “A Genetic Algorithm Tutorial”, (November 1993).
11) Donald H.Kraft, Frederick E. Petry, Bill P. Buckles, Thyagarajan Sadasivan, “The Use of Genetic Programming to Build Queries for Information Retrieval”, IEEE, pp 468-473,(1994).
12) E.Damiani And M.G. Fugini, “Fuzzy Techniques for Software Reuse” Association For Computing Machinery, Inc, (ACM)”, pp 552-557, (1996).
13) E.Damiani And M.G. Fugini, “Design and Code Reuse Based on Fuzzy Classification of Components”, ACM, pp26-32, (1996).
14) E.Damiani, And M.G. Fugini, "Automatic Thesaurus Construction Supporting Fuzzy Retrieval of Reusable Components", Proc. ACM SIG-APP Conf. (SAC'95), Nashville, (Feb. 1995).
15) Faisal Siddiqui “CBSE: A Look at Reusable Software Components”, (1999).
16) Francesco Baruchelli and Giancarlo Succi, “A Fuzzy Approach to Faceted Classification and Retrieval of Reusable Software Components”, pp 15-20, (1997).
17) Giancarlo Succi, Carl Uhrik, Tullio Vernazza, “A Formal View to Classification and Retrieval Mechanism for Reusable Objects”, pp 27-32.
18) Guohong Zhang, “Component-Based Software Engineering”, Thesis, (2000).
19) J M Spivey, “An Introduction to Z and Formal Specifications” (January 1989).

20) Jeffrey G. Gray, “Research Issues in Characterizing the Performance of Reusable Software Components”, (October 1995).
21) Jorng-Tzong Horng, Ching-Chang Yeh, “Applying Genetic Algorithms to Query Optimization in Document Retrieval”, Information Processing and Management, 36, pp 737-759, (2000).
22) Ju An Wang, “Towards Component-Based Software Engineering”, (November 2000).
23) Jun-Jang Jeng, Betty H. C. Cheng, “Specification Matching for Software Reuse: a Foundation”, ACM, pp 97-105, (1995).
24) Larry Smith, “Software Reuse Technologies and Applications”, (December 1994).
25) M.R. Girardi And B. Ibrahim, “An Approach to Improve the Effectiveness of Software Retrieval”, (1993).
26) M.R. Girardi And B. Ibrahim, “Automatic Indexing of Software Artifacts”, (1994).
27) Mehrdad Dianati, Insop Song, Mark Treiber, “An Introduction to Genetic Algorithms and Evolution Strategies”.
28) Michael Gordon, “Probabilistic and Genetic Algorithms for Document Retrieval”, Communications of the ACM, Volume 31, Number 10, pp 1208-1218, (1988).
29) Oualid Khayati, Jean-Pierre Giraudin, “Components Retrieval Systems” (2001).
30) Panos Constantopoulos and Martin Dorr, “Component Classification in the Software Information Base”, Chapter 7, pp. 177-200, Prentice Hall, (1995).
31) Paul V. Biron, Donald H. Kraft, “New Methods for Relevance Feedback: Improving Information Retrieval Performance”, ACM, pp 482-487, (1995).
32) Praveen Pathak, Michael Gordon, Weiguo Fan, “Effective Information Retrieval Using Genetic Algorithms Based Matching Functions Adaptation, Proceedings of the 33rd Hawaii International Conference On System Sciences”, pp 1-8. (2000).
33) Qin He,” Neural Network and its Application in IR”, (1999).
34) Robert J. Hall, “Generalized Behavior-Based Retrieval”, (1993).
35) Rubén Prieto-Díaz, “Software Reuse: Issues and Experiences”, American Programmer vol.6, No.8, pp 10-18, (April 1993).
36) Rubén Prieto-Díaz, “Implementing Faceted Classification for Software Reuse”, Communications of the ACM”, Vol. 34, pp 88-97, (1991).
37) Rune Meling, E James Montgomery, Pon Sudha Ponnusamy, Eva Beverly Wong, Daniela Mehandjiska, “Storing and Retrieving Software Components: A Component Description Manager”.
38) Sathit Nakkrasae, Peraphon Sophatsathi, William R.Edwards, “Fuzzy Subtractive Clustering Based Indexing Approach for Software Components Classification”.
39) Stephanie Forrest, “Genetic Algorithms”, ACM Computing Surveys, Vol. 28, No. 1, pp 77-80 (March 1996).
40) Steven Atkinson, “Examining Behavioural Retrieval”.

41) Weiguo Fan, Michael D. Gordon, Praveen Pathak, "A Generic Ranking Function Discovery Framework By Genetic Programming for Information Retrieval",Information Processing And Management, pp 587–602, (2004).
42) Weiguo Fan, Michael D. Gordon, Praveen Pathak, "Automatic Generation of a Matching Function by Genetic Programming for Effective Information Retrieval", (1998).
43) Zhu Jingbo, Yao Tianshun "A Knowledge-Based Approach to Text Classification" (2001).

List of Papers

1. Navneet Kaur, Rajesh K. Bhatia, "Retrieval of Best Component using Genetic Algorithms" communicated to IEEE/ACM International Conference on Automated Software Engineering, USA, to be held in Nov 2005.

2. Navneet Kaur, Rajesh K. Bhatia, "Information Retrieval from a Component Based Repository Using Genetic Algorithms" communicated to Indian International Conference on Artificial Intelligence, Pune, to be held in Dec 2005.