MOBILE AGENTS IN REUSABLE SOFTWARE COMPONENT RETRIEVAL

A thesis
Submitted in the partial fulfillment of the
requirement for the award of degree
of

Master of Engineering
in
Software Engineering

Under the Supervision of
Mr. Rajesh K. Bhatia
Assistant Professor,
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala.

Submitted By
Amandeep Singh
(8023101)

–

Computer Science & Engineering Department
Thapar Institute Of Engineering & Technology
(Deemed University), Patiala-147004 (India)

May 2004

Abstract

Software is becoming part of almost every operation of the modern world. Competition is becoming both intense and more widespread as more and more organizations depend on mission-critical business information systems to manage customer interactions, speed goods to their destinations, and manage finances in an ever more complex, global business environment. Often companies compete on small differences in quickly introduced, innovative services. Improving business performance often means they must dramatically improve their software development performance. Building software systems from previously developed, high-quality components certainly saves the cost and time of redundant work and improves systems.

Dedicated centralized search engines, such as Yahoo or Google, carry out currently all searching on the Internet and large networks. Developing such systems tends to be extremely expensive in terms of hardware, bandwidth requirements, and also the specialized algorithms and software that are necessary. Most Internet search engines maintain a very large centralized database, which is updated by crawling the Internet and indexing websites. When a query is received the database replies with a list of websites that are deemed to be in some way related to the original query.

So while current search systems may be suitable for general web searching they do have disadvantages. The current size of the Internet, and the rate at which it is growing, makes it impossible to visit every website and index it. Indexing only works well on websites that contain static information, as the search engine may not visit the website regularly enough to be able to index new content.

Mobile agent paradigm has the capability to send a piece of software anywhere on the Internet, which can then act autonomously and can react to various situations. Mobile agents can be used in Electronic Markets, Distributed Information Retrieval Systems, Remote Device Control and Configuration etc.

A database search system for Internet using mobile agents, which accepts query in the form of formal specification in $L^{A}T_{E}X$ format and presents query results as percentage match for each component has been proposed in the current work. This work can be extended further using some other possible design patterns.

Declaration

I hereby certify that the work which is being presented in the thesis entitled, "***Mobile Agents in Reusable Software Component Retrieval***", in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work 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.

Amandeep Singh

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

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

Countersigned by

(Seema Bawa)
Assistant Professor & Head,
Computer Sc. & Engg. Department,
Thapar Institute of Engg. & Technology,
Patiala.

(Dr. D.S. Bawa)
Dean (Academic Affairs)
Thapar Institute of Engg. & Technology,
Patiala.

The M.E. (Thesis) Viva-Voice examination of Amandeep Singh, Roll No. 8023101, M.E. (Software Engineering), Thapar Institute of Engineering and Technology, Patiala has been held on. …….

Supervisor **External Examiner**

Acknowledgement

I am very thankful to the Department of Computer Science & Engineering, Thapar Institute of Engineering and Technology, who has given me a chance to work on this thesis and provided me all types of resources when needed.

No amount of words can adequately express the debt, I owe to Mr. Rajesh Kumar Bhatia, Assistant Professor, Computer Science & Engineering Department, for his kind support, motivation and inspiration that triggered me for the thesis work. I owe him lots of gratitude for having me shown this way of research. He could not even realize how much I have learned from him.

I wish to express my gratitude to Ms. Seema Bawa, Assistant Professor & Head, Computer Science & Engineering Department, for their excellent guidance and encouragement right from beginning of this course. I am also thankful to all the faculty and staff members of the Computer Science & Engineering Department for providing me all the facilities required for the completion of this work.

Most importantly, I would like to give God the glory for all of the efforts I have put into this report.

Amandeep Singh
Roll no. 8023101

Contents

List of Figures and Tables

Figures

Tables

Organization of thesis

Thesis is organized in four main parts, covering the Details of Mobile agents along with details of network computing paradigms, mobile agent applications and properties of

mobile agents, idea and details of software reuse, component retrieval techniques and a model proposed for reusable software component retrieval.

Chapter 1 provides the basic introduction about network computing paradigms, the Mobile agents, its Properties and Application areas.

Chapter 2 gives the overview of software reuse, current trends and various component retrieval techniques.

Chapter 3 describes the proposed model for retrieving reusable software components.

The work done has been concluded along with future directions in Chapter 4.

Chapter 1
Introduction

New technologies are required to face new scenarios due to the fast spread of the Internet. The Technologies, architectures, and methodologies traditionally used to develop distributed applications exhibit a variety of limitations and drawbacks when applied to large scale distributed settings e.g. Internet. In particular, they fail in providing the desired degree of configurability, scalability, and customizability. To address these issues, researchers are investigating a variety of innovative approaches. The most promising and intriguing ones are those based on the ability of moving code across the nodes of a network, exploiting the notion of mobile code. Mobile agent is a revolutionary alternative to RPC, inspired by the highly distributed and heterogeneous computation environment for Internet. In RPC systems, processes cooperate with each other through remote communication. On the contrary in mobile agent systems agents move to remote host and then communicate with target processes and access desired resources locally. Even though in terms of network communication RPC and agent mobility are both realized by passing data remotely, the biggest difference between them is that the agent mobility transports not only ordinary static values but also program codes to execute the agent on destination sites.

Mobile agents promise to bring in a new era in the field of World Wide Web and Internet Computing. Although mobile agents have been around for sometime, their full potential has not been realized due to the lack of a suitable infrastructure that would allow for their deployment and seamless integration on the Internet.

1.1 Agent

Franklin and Graesser proposed a definition for autonomous agents: *An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.* This definition attempts to capture the essence of being an agent, and to define the broadest class of agents.

1.1.1 Software Agent

Software agents [7] are programs that assist people and act on their behalf. Agents work by allowing users to delegate work to them. They live in an environment and have the ability to asynchronously and autonomously interact with their execution environment. Agents do not require anyone either to deliver information to agent or to consume any of its output. In order to complete it's own goals, agent acts continuously.

Stationary agents are a kind of agents, which sit at a place and communicate to its environment by conventional means e.g. remote procedure calling and messaging. A stationary agent executes only on the system where it begins execution. Communication mechanism such as Remote Procedure Calling (RPC) is used to get information stored in some other system or to interact with an agent on a different system, if needed.

1.2 Network Computing Paradigms

Various Network Computing Paradigms are as follows:

- Client Server Paradigm
- Code on Demand Paradigm
- Mobile Agent Paradigm

1.2.1 Client Server Paradigm

In this Paradigm [7], resources of the server such as database are accessed by the client through a set of services advertised by the server. The server hosts code implementing this set of services locally.

We say that the server holds the *know-how*. Finally, it is the server itself that executes the service and thus has the *processor* capability as shown in Figure 1.1. For accessing a resource hosted by the server, the client uses one or more of the services provided by the server.

Some intelligence is required by the client to decide which of the services it should use. The server has it all: the know-how, resources, and processor. Most distributed systems have been based on this paradigm as it is supported by a wide range of technologies such as remote procedure calling, object request brokers (CORBA), and java remote method invocation (RMI).

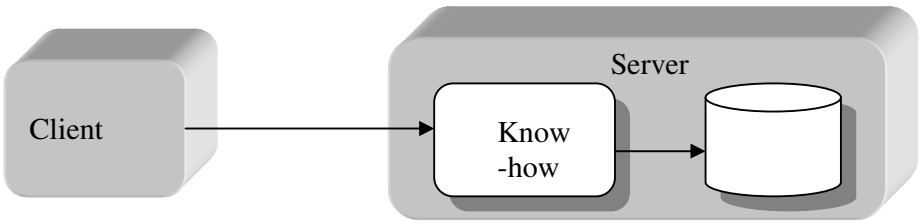


Fig. 1.1: Client-Server Paradigm

1.2.2 Code on Demand Paradigm

In this Paradigm [7], the server whenever needed provides code to the client. The client gets the know-how whenever needed.

Suppose that a client initially is unable to execute its task because of lack of code i.e. know-how and a host in the network provides the needed code. Once the client

receives the code, the computation is carried out in the client. The client holds the processor capability as well as the local resources. In contrast to the classical client-server paradigm, the client does not need preinstalled code because all the necessary code will be downloaded. So the client has the resources and processor and the host has the know-how as shown Figure 1.2.

Some of the practical examples of this paradigm are Java applets and servlets. Applets get downloaded in web browsers and execute locally, whereas servlets get uploaded to remote web servers and execute there.

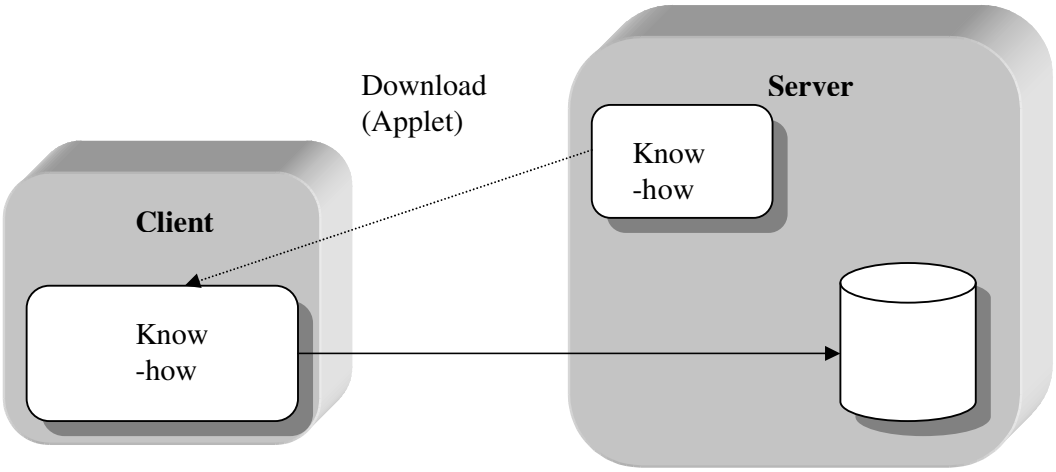


Fig 1.2: Code-on-Demand Paradigm

1.2.3 Mobile Agent Paradigm

In this Paradigm [7,21], any host in the network is allowed a high degree of flexibility to possess any mixture of know-how, resources, and processors. Its processing capabilities can be combined with local resources. Know-how in the form of agents is not tied to a single host but rather is available throughout the network as shown in Figure 1.3.

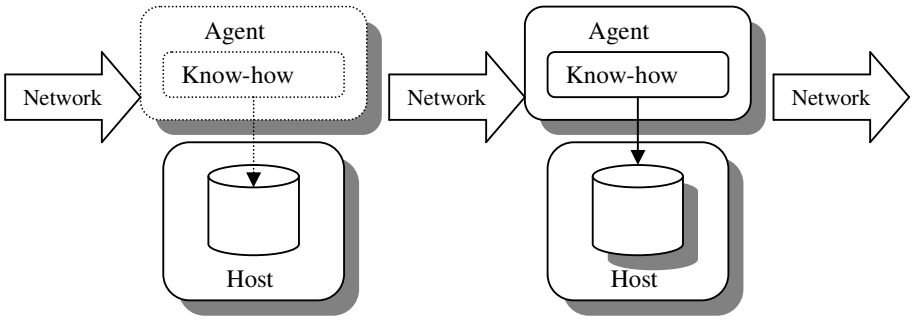


Fig 1.3: Mobile Agent Paradigm

1.3 Mobile Agent

A mobile agent [5,7] is a running program that can move from host to host in a network when and where it chooses. Mobility is an orthogonal property of agents, i.e. not all agents are mobile.

A mobile agent is not bound to the system where it begins execution and has the unique ability to transport itself from one system in a network to another. The mobility of an agent allows it to move to a system that contains an object with which the agent wants to interact and then to take advantage of being in the same host or network as the object. A mobile agent can transport its state as well as its code from one execution environment to another in a network.

By the term state, we typically mean the attribute values of the agent that help it determine what to do when it resumes execution at its destination. By the term code, we mean, in an object-oriented context, the class code necessary for the agent to execute.

Information is being disseminated at every server that the mobile agent visits. Every server benefits from accepting a visiting mobile agent, because the mobile agent will have either new or more recent information about resources. Also, every mobile agent benefits from visiting a peer because it will learn of either new or updated resources. If the mobile agents do not contain any new information they may be destroyed. Accepting and

hosting mobile agents requires the use of physical resources, such as memory and computer cycles. If resources are critically limited, it is easy for the server to refuse further requests to accept mobile agents until more physical resources become available.

Mobile agents may easily be cloned and dispatched in different directions. This allows them to function in parallel. Although this causes more mobile agents to be active on the network, it does ensure that the network resource discovery is completed sooner, and therefore the mobile agents spend less time on the network.

A mobile agent based solution is very fault tolerant. Even if some of the mobile agents are destroyed, all surviving ones will have a positive impact. Indeed, the destroyed mobile agents will have benefited every peer up to the point where they were destroyed.

1.3.1 Properties of mobile agents

Following are some of the *Compulsory Properties* of an agent: -

- *Autonomous*: - It is capable of controlling it's own actions depending upon its surrounding environment.
- *Communicative*: -Agent has the availability to exchange messages with other agents or systems. This feature is also known as social agent.
- *Goal-driven*: **-** This property is defined that agent does not simply act in response to the environment, but also from information it has in its model or it builds it-self. That means the goal is planed ahead. This availability is named also pro-active purposeful.
- *Reactive*: - It senses changes in the environment and acts in accordance with those changes. They perceive their environment consisting of humans, agents as well as arbitrary objects and react on it.
- *Temporally continuous*: **-** Continuous execution is in its nature.

Following are some of the *Orthogonal Properties* of an agent: -

- *Mobile*: **-** It can travel from one host to another
- *Strong Mobility*: Ability of a system to migrate code and execution state of execution unit.
- *Weak Mobility*: Ability of a system to migrate code of execution unit.
- *Learning*: - It adapts in accordance with previous experience.
- *Believable:* - It appears believable to the end-user.

1.3.2 Advantages of mobile agents

There are a lot of reasons to use mobile agents, some of them are listed here: -

- They reduce the network load. Distributed systems often rely on communication protocols that involve multiple interactions to accomplish a given task. This is especially true when security measures are enabled. The result is a lot of network traffic. Mobile agents allow us to package a conversation and dispatch it to a destination host where the interactions can take place locally as shown in Figure 1.4. Mobile agents are also useful when it comes to reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data, rather that transferred over the network. The motto is simple [7]: *move the computations to the data rather than the data to the computations*.

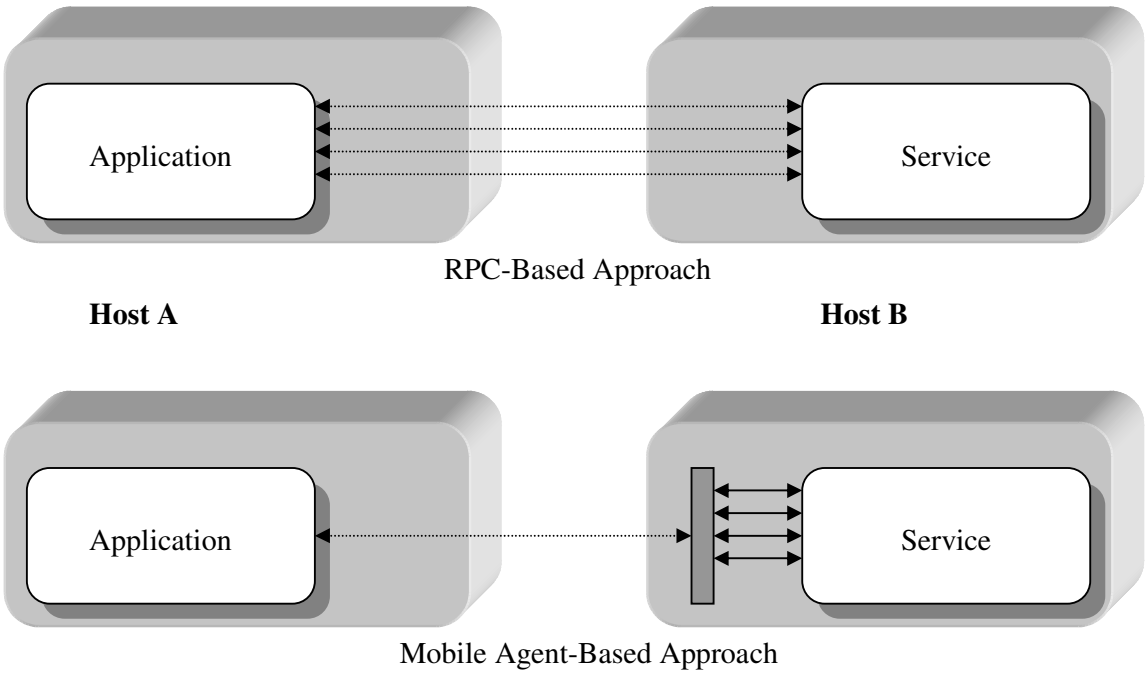


Fig. 1.4: Mobile Agents and Network Load Reduction

- They overcome network latency. Critical real-time systems such as robots in manufacturing processes need to respond to changes in their environments in real time. Controlling such systems through a factory network of a substantial size involves significant latencies. For critical real-time systems, such latencies are not acceptable. Mobile agents offer a solution, since they can be dispatched from a central controller to act locally and directly execute the controller's directions.
- They encapsulate protocols. When data are exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data, respectively. However, as protocols evolve to accommodate new efficiency or security requirements, it is a cumbersome if not impossible task to upgrade protocol code properly. The result is often that protocols become a legacy problem. Mobile agents, on the other hand,

are able to move to remote hosts in order to establish "channels" based on proprietary protocols.

- They execute asynchronously and autonomously. Often mobile devices have to rely on expensive network connections. That is, tasks that require a continuously open connection between a mobile device and a fixed network will most likely not be economically or technically feasible. Tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the mobile agents become independent of the creating process and can operate asynchronously and autonomously. The mobile device can reconnect at some later time to collect the agent as shown in Figure1.5.

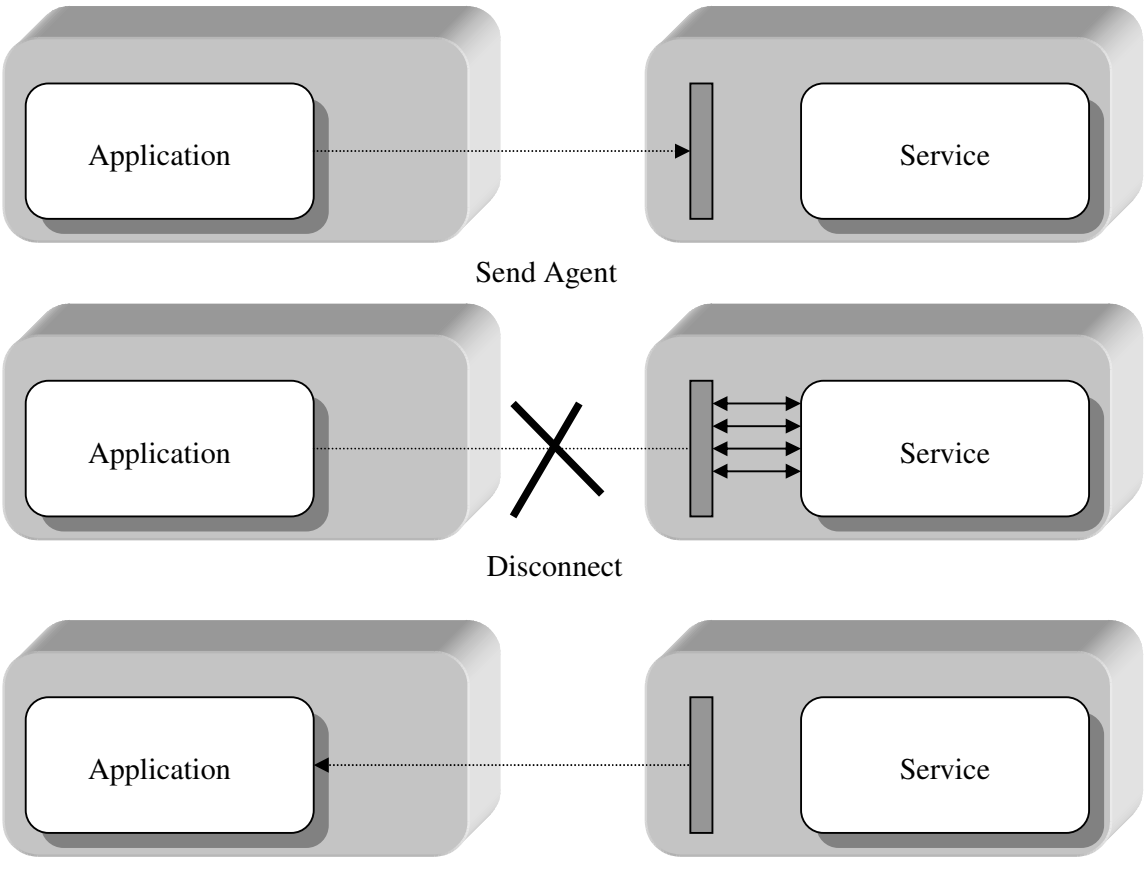


Fig. 1.5: Mobile agents and Disconnected Operations

- They adapt dynamically. Mobile agents have the ability to sense their execution environment and react autonomously to changes. Multiple mobile agents possess the unique ability to distribute themselves among the hosts in the network in such a way as to maintain the optimal configuration for solving a particular problem.
- They are naturally heterogeneous. Network computing is fundamentally heterogeneous, often from both hardware and software perspectives. As mobile agents are generally computer- and transport-layer-independent, and dependent only on their execution environment, they provide optimal conditions for seamless system integration.
- They are robust and fault-tolerant. The ability of mobile agents to react dynamically to unfavorable situations and events makes it easier to build robust and fault-tolerant distributed systems. If a host is being shut down, all agents executing on that machine will be warned and given time to dispatch and continue their operation on another host in the network.

1.3.3 Applications of mobile agents

Following are some of the areas where mobile agents can be used: -

- Electronic Markets: - Because of properties as listed above, mobile agents fit quite well in representing participants of electronic markets [9,25] and their policies. Agents may sell their own services to users, hosts, and other agents. In electronic markets they can perform a lot of time consuming but simple tasks for their users (e.g. searching for cheap supplies).
- Distributed Information Retrieval: - An agent based solution for distributed information retrieval provides one or more agents that visit WWW servers searching for interesting pages. This approach saves bandwidth because the only

needed communications are those to send and receive the agent. This solution also fits well in mobile computing [9,25], in a scenario where a PDA is only occasionally connected to the Internet, an user can send his searching agent, disconnect and reconnect later, when the agent has finished his job and comes back to the user to show the results of the query.

- Parallel Computing: Solving a complex problem on a single computer takes a lot of time. To overcome this, mobile agents can be written to solve the problem. These agents migrate to computers on the network, which have the required resources and use them to solve the problem in parallel thereby reducing the time required to solve the problem.
- Data Collection: Consider a case wherein, data from many clients has to be processed. In the traditional client-server model, all the clients have to send their data to the server for processing resulting in high network traffic. Instead mobile agents can be sent to the individual clients to process data and send back results to the server, thereby reducing the network load.
- E-commerce: Mobile agents can travel to different trading sites and help to locate the most appropriate deal, negotiate the deal and even finalize business transactions on behalf of their owners. A mobile agent can be programmed to bid in an online auction on behalf of the user. The user himself need not be online during the auction.
- Mobile Computing: Wireless Internet access is likely to stay slow and expensive. Power consumption of wireless devices and high connection fee deter users from staying online while some complicated query is handled on behalf of the user.

Users can dispatch a mobile agent, which embodies their queries, and log off, and the results can be picked up at a later time.

1.4 Summary

Mobile agents are a promising solution for design and implementation of large scale distributed applications, since it overcomes many drawbacks of the traditional client server approach. There is a clear evolutionary path that will take us from current technology to widespread use of mobile code and agents within the next few years. Once several technical challenges have been met, and a few pioneering sites install mobile agent technology, use of mobile agents will expand rapidly.

Chapter 2
Software Reuse

Software reuse has been recognized as an attractive idea with an obvious payoff since Doug McIlroy first proposed libraries of shared components in 1968. Reduction in Cost and time of redundant work, system improvement can be done by building software systems from previously developed, high-quality components.

For many years, obtaining high levels of reuse has been elusive because many different technical, process, and organizational issues have blocked progress. But despite pursuit of a variety of other methods to improve software development, it remains clear that systematic software reuse and component-based development is still one of the most promising ways to significantly improve the productivity of software development.

A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A lot of research on how to reuse already made and tested software components has been done because of the need for higher productivity in software development. As libraries of reusable software components continue to grow, the issue of retrieving components from software libraries has captured the attention of the software reuse community.

Reusable components are normally identified by the characteristics of their interfaces. That is, “the services that are provided, and the means by which consumers access these services” are described as part of the component interface. But the interface does not provide a complete picture of the degree to which the component will fit the architecture and requirements. The software engineer must use a process of discovery and analysis to qualify each component’s fit.

2.1 Component

A reusable software Component [27] can be described in many ways, but an ideal description is called the 3C model- *concept, content, and context*.

The *concept* of a software component is a description of what the component does. The interface to the component is fully described and the semantics-represented within the context of preconditions and post conditions- are identified. The concept should communicate the intent of the component.

The *content* of a component describes how the concept is realized. In essence, the content is the information that is hidden from casual users and need be known only to those who intend to modify or test the component.

The *context* places a reusable software component within its domain of applicability. That is, by specifying conceptual, operational, and implementation features, the context enables a software engineer to find the appropriate component to meet application requirements.

2.2 Component Reuse

Component based software engineering (CBSE) [27] is a process that emphasizes the design and construction of computer based systems using reusable software components. CBSE encompasses two parallel engineering activities: domain engineering and component based development as shown in Figure 2.1. Domain engineering explores an application domain with the specific intent of finding functional, behavioral, and data components that are candidates for reuse. These components are placed in reuse libraries. Component based development collects requirements from the customer, selects an appropriate architectural style to meet the objectives of system to be built, and then

1. Selects potential components for reuse

2. Qualifies the components to be sure that they properly fit the architecture for the system
3. Adapts components if modifications must be made to properly integrate them
4. Integrates the components to form subsystems and the application as a whole.

In addition custom components are engineered to address those aspects of the system that cannot be implemented using existing components.

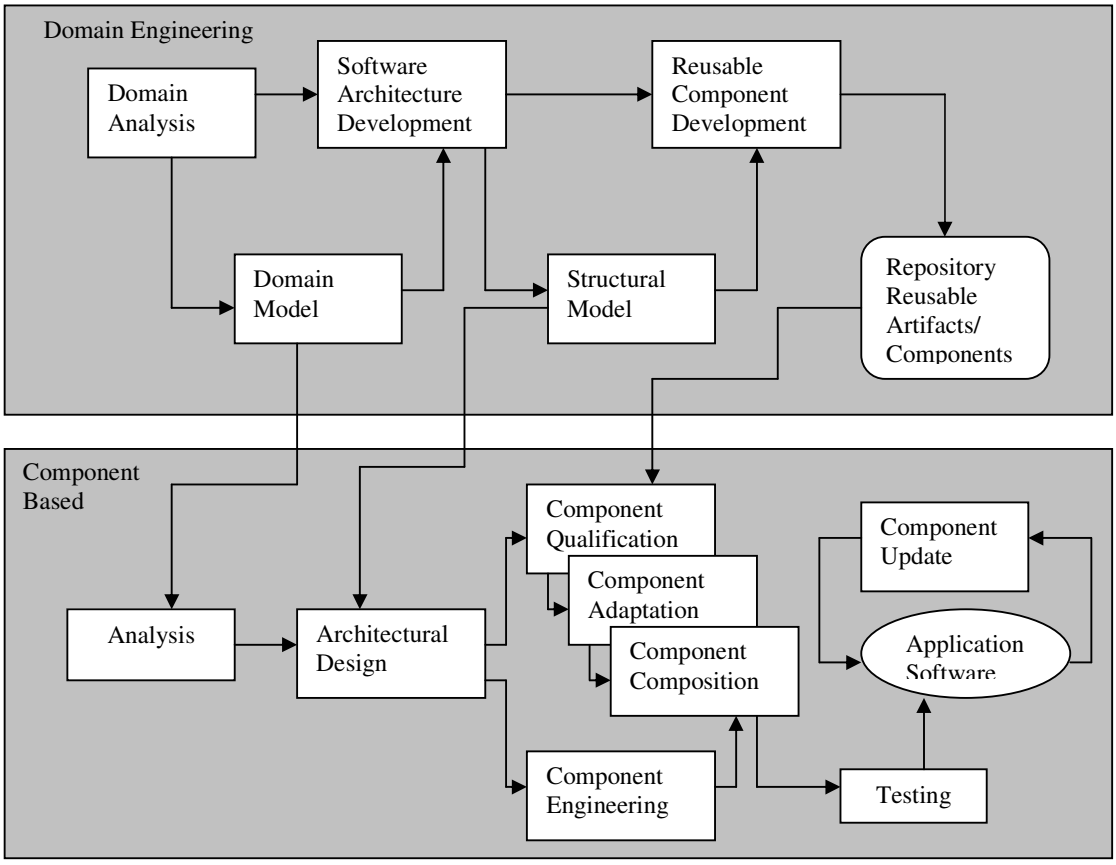


Fig. 2.1: Component based software engineering

Transforming software requirements into a software design involves the iterative partition of a solution into software components. The partition process starts with the identification of basic system and sub-system at high-level design components and concludes with the identification of module at low-level design components to be implemented such as modules, packages, and library specifications.

Component-based approaches for the development of information systems are a widely growing engineering discipline that deals with the cycle of developing components and developing with components. Software engineers try to borrow concepts of components composition from other engineering disciplines to component software. Since the early nineties when Microsoft introduced a component-based development environment (the Visual Basic environment and its plug gable components), several component-based approaches and components models were introduced (COM, DCOM, EJB, CORBA,...). Component-based development process is composed of two complementary sub-processes:

-*Applications engineering sub-process*: application engineers use components to build applications.

-*Components engineering sub-process*: component engineers identify, develop and capitalize components that have a high usability and usefulness.

Components libraries are so used by different actors and the real actor number depends on the development team organization Component library management is the central feature of any component-based development approach. This work aims to propose a new component retrieval approach by combining existing approaches and to put into advance the context in which they can really be helpful for applications engineers.

2.3 Reuse Metric

A variety of software metrics [27] have been developed in an attempt to measure the benefits of reuse within a computer based system. The benefit associated with reuse within a system S can be expressed as a ratio

$$R_b(S) = [C_{noreuse} - C_{reuse}]/C_{noreuse}$$

Where

$C_{noreuse}$ is the cost of developing s with no reuse.

C_{reuse} is the cost of developing S with reuse.

2.4 Retrieval Techniques

Following are some of the classification and retrieval techniques:

- Classification schemes
- Automatic indexing
- Formal specifications
- Knowledge based approaches
- Behavior based retrieval
- Neural network based retrieval
- Browsing
- Hypertext

2.4.1 Classification schemes

Main approaches for software classification [30] and retrieval propose a classification scheme for cataloguing components in a software base. The schemes specify some attributes (called facets) to be used as descriptors of a software component, mostly focusing on the action that a component performs and on the objects manipulated by the component. Specifying keywords for each attribute in the scheme performs both classification and retrieval. Term relationships or phrases are expressed through attribute combination, allowing then better precision than simple keyword systems. These approaches are based on a controlled vocabulary, usually derived from Domain Analysis, which establishes a limited set of terms or phrases that the user or indexer has to select manually for classification or retrieval activities. The creation of such a controlled

vocabulary is labor intensive and therefore expensive. On the other hand, retrieval under a controlled vocabulary normally requires an important effort from untrained users, which could be avoided with a free vocabulary or natural language specifications.

2.4.2 Automatic indexing

Free text indexing systems automatically extract keywords from queries in natural language and these keywords are used to locate software components. Similarly, software components are classified in a software base by indexing [12] them according to keywords extracted from the natural language documentation of the software components. Most of these systems work at the lexical level, so they do not use the syntactic and semantic information available in a description in natural language. Because terms are extracted automatically from software descriptions, indexing is cheaper than indexing approaches based on a controlled vocabulary. Also, since the vocabulary is uncontrolled, indexing can be done as specific as needed. These systems use information retrieval statistical techniques for automatic indexing which require the existence of a large corpus of text to provide acceptable retrieval effectiveness. Unfortunately, software descriptions not always satisfy this requirement, limiting the effectiveness of these techniques when applied to software retrieval. On the other hand, indexing terms do not capture the semantic information associated to the functionality of a software component.

2.4.3 Formal specifications

In this approach, queries are formal requirement specifications [11] e.g. the specification of a system 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.

Advantage of this approach is the fact that formal specifications focus on the behavior of the component rather than on descriptive information, that they are free from ambiguity and that they provide better precision than informal methods. Disadvantages of this approach are: -

- Formal specifications constitute a very small proportion of the software available for reuse.
- Processing times for the search algorithms may be excessive depending on the approach taken.
- It is easier and cheaper to use textual descriptions as queries than formal specifications.
- Partial matching on retrieval by similarity is difficult to control because these approaches are more based on exact matching.

If formal software development becomes popular enough to justify the investment in the construction of software bases of formal specifications, these specifications should be the main source of information for indexing activities. Not only because this information comes directly from the object to be located and not from a software document that can describe inappropriately the software component, but also because it appears to be a natural way to determine the differences between a searched component and a component in the software base in order to estimate the reuse effort for adapting the component. The main problem is that current approaches for retrieval through formal specifications use existent formalisms, which are maybe not appropriate enough to provide good information for indexing purposes and make their automatic extraction possible. It should perhaps be the other way around, i.e. adapting existing formalisms or defining new ones to take into account the retrieval and classification

2.4.4 Knowledge based approaches

Knowledge based software retrieval systems [12] make some kind of lexical, syntactic and semantic analysis of natural language specifications of software components without pretending to completely understand the documents. They are based on a knowledgebase, which stores semantic 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.

2.4.5 Behavior based retrieval

These methods aim at exploiting the execution ability of software components. Executing software components in a software base according to the input provided by the user and comparing the resulting output with the one supplied by the user perform retrieval. These approaches can be effective if users have a pre-established set of test cases when submitting their queries, which, unfortunately, is not always the case.

2.4.6 Neural network based retrieval

Neural Network [12] based methods uses artificial neural network technology to structure software bases according to the functional similarity of the stored software components, i.e. components that exhibit similar behavior are stored near to each other. Thus, the similarity between components is made explicit because it is determined according to the Software retrieval based on natural language specifications geographical closeness between components. Neural networks provide a suitable framework for incrementally adapting the conceptual distance weight based upon user's relevance assessments.

Single term indexing techniques are used to extract from both user's queries and software descriptions the keywords used by the learning algorithm of the artificial neural network in classification and retrieval activities. Neighborhood functions used by the learning algorithm are based on traditional measures of similarity of the vector space model.

2.4.7 Browsing

Most of the approaches described above propose linear retrieval as the main retrieval mechanism, i.e. a mechanism for retrieving a set of potentially reusable components from a user's specification of the required component. Some of these approaches also include browsing as a complementary retrieval mechanism, where a starting point in the search space is determined from the user's query. Pure browsing approaches have also been proposed, like the Smalltalk-80 System Browser, which is based on a four level hierarchy of structured navigation. The main disadvantage of these tools is that they require the user to drive the navigation process, which can be difficult when having large software bases.

2.4.8 Hypertext

In hypertext systems the information is organized as a network of nodes or information units that are interconnected according to different links or relationships. Users navigate through the information network by following existing links and the semantics associated with the links can guide users to appropriate nodes in their information search. The main problem of these approaches is that developing and maintaining a software base in a hypertext environment require a considerable amount of human effort. Frequently, there are no good guidelines to abstract semantic relationships to link software components and, in order to include a new component in the software

base, it is necessary to consider all possible links between the new component and the components already present in the software base.

2.5 Summary

Component based software engineering offers inherent benefits in software quality, developer productivity, and overall system cost. 3C component model has been discussed in this chapter. Various component retrieval techniques are also explored. A new agent based component retrieval model is considered in next chapter.

Chapter 3
Agent Based Proposed Model

Keyword-based search paradigm has been popularized by many search engines on World Wide Web, where a user can specify a string of keywords and expect to retrieve relevant documents, possibly ranked by their relevance to the query. Since a lot of information is stored in reusable component databases and not as HTML documents, it is important to provide a similar search paradigm for such databases, where users can query a database without knowing the database schema and database query languages such as SQL. A Distributed Component Repository search system for Internet or intranet using mobile agents, has been proposed in the current work, which accepts query in the form of formal specification in $L^{A}T_{E}X$ format and presents query results as percentage match for each component.

Mobile agent paradigm [1] has the capability to send a piece of software anywhere on the Internet, which can then act autonomously and can react to various situations. Software reuse is the requirement of the day as developing a new system from the scratch is too much costly and time consuming than reusing the components. So combining the two approaches i.e. Mobile agent Paradigm and Component retrieval approach, a hybrid model for searching software components on the Internet is developed.

In this chapter first component, problem of searching components and a mobile agent based model for searching and retrieving from the repository are discussed.

3.1 Current Trend

Nowadays, as a result of the growing use of the Web and networks, the complexity of software systems is increasing. Due to the heterogeneous nature of the distributed

environments where they should run, these systems need to model new kind of interactions, and also they may be able to be adapted to changing requirements and new trends. In order to cope with this complexity we need new paradigms that facilitate the design of distributed and open systems in a better way than the object-oriented paradigm.

Having into account actual running trends of the software engineering, two paradigms are applicant for this purpose, the component and agent-based paradigms. Both of them seem to represent an evolution of the object-oriented paradigm. However, each paradigm focuses on different aspects of distributed applications and so they might be complementary.

The mobile agent paradigm [9] is especially attractive to perform complex, tedious or repetitive tasks in open and dynamic systems. Agent-based applications are developed specially for dynamic, distributed, open and heterogeneous environments, such as the Internet. The basic entity for designing and building this kind of applications is the agent, which can be seen at some scale, like an active object. Therefore, the mobile agent paradigm may represent another step in the evolution of the component reuse.

Getting a look to the literature about components, we cannot find a consensus about what are the characteristics that a component might offer, but there are some of them that are present in almost every component definition. *Introspection* lets components to 'say something' about their behavior and data. This property, which is not found on current agent systems, would allow agents to interoperate dynamically, and it would let them to afford, for example, a protocol based on introspection.

Components are passive entities that model resources and real world entities like a camera, a printer, or a database. The Interface Definition Language (IDL) of a component defines the list of messages that can be received or sent by the component. The reception of a message causes changes in the internal state of components that are reported to the

environment by sending one or more messages specified as part of its IDL. This means that an IDL does not only include information about the incoming messages as usual and also the outgoing ones.

Current work enforces to share and reuse components using mobile agents. To make easier this task, the platform offers a Distributed Component Repository (DCR). In this way, application designers may reuse public software developed by others companies.

Following Web principle of information distribution, every server machine has a Component Repository, organized in hierarchical domains. Each entry of the repository corresponds to an implementation of a component, and contains some information needed to know component. The information available for a component is its IDL and a brief textual description of its behavior.

Classification and selection of reusable components are considered as a key success factor. Software component suppliers should provide standard documentation, to help customers in finding quickly the right component.

3.2 Reusable Components

Component Based Reuse (CBR) - creating applications from reusable parts - has been gaining more interest within the software developers community. Component models such as JavaBeans, ActiveX have a great impact on CBR. Software developers can rapidly and visually construct applications by assembling reusable JavaBeans or ActiveX components by visual application builder tools. For example, JavaBeans components can be created and assembled by using visual application builder tools such as IBM's VisualAge for Java, Borland's JBuilder, Symantec's Visual Cafe, SunSoft's Java Workshop, Sysbase's PowerJ and many others.

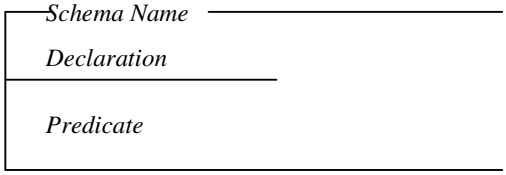
The growth in the usage of Internet forces us to re-evaluate CBR. Today it is possible to produce a reusable component that runs in any kind of platform using the JavaBeans technology. It is also possible to download these reusable components from Internet and assemble them using visual application builder tools. It can be expected that in a very near future many academic, governmental and commercial organizations will be marketing the components that they produced on Internet. Therefore, Internet with the components containing analysis models, design patterns, Java Beans codes, Active-X codes, documentations, etc. will become the component repository of any organization.

3.3 Formal Specifications

Formal methods [19,28] are used to create a specification that is more complete, consistent, and unambiguous than those produced using conventional or natural specifications. Formal specification is a concise description of the behavior and properties of a system written in a mathematical-based language.

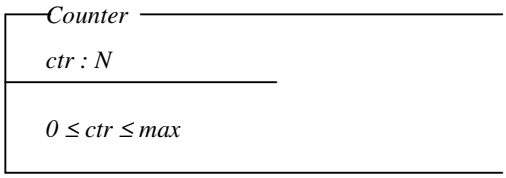
Z is a formal specification language, which is based on set theory and predicate logic. It was developed originally at Programming Research Group, Oxford University. It is declarative, not procedural. Introducing fixed sets and variables and specifying the relationships between them using predicates describe the system. The operations are expressed by relationship between values of variables before, and values after, the operation.

The basic unit of specification in Z is a schema. A Z schema consists of a name, a declaration of variables, and a predicate.



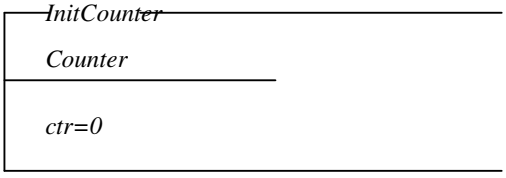
A system specification in Z consists of some state variable, an initialization, and set of operations on the state variables. The state variables will also have some invariants associated with them representing "healthiness conditions" which must always be satisfied. Usually all of these are specified using schemas.

For example, the state variables of a counter system may be specified using the following schema:

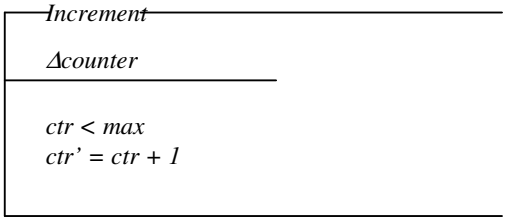


Here, ctr is declared to be a natural number and the predicate part describes an invariant that must be satisfied by ctr, the state variable of the system.

An initialization may be specified as follows:



An operation is specified in Z with a predicate relating the state before and after the invocation of that operation. For example, an operation to increment the counter may be specified as follows:



The declaration *Δcounter* means that the state *Counter* is changed by the operation. Similarly, Ξ*Counter* means that the operation cannot change the state of counter, so *ctr'* = *ctr*. In the predicate, the new value of a variable is primed (*ctr'*), while the old one is unprimed. So the above predicate states that the new value of the counter, *ctr'*, is the old value plus one. There is an implicit conjunction (logical-and) between successive lines of the predicate part of a schema.

An operation may also have input and output parameters. Input parameter names are usually suffixed with '?', while output parameters names are suffixed with '!'. Terminology of Z notation is listed in Appendix I.

3.4 Latex Keywords

Parsing the concrete representation of a Z specification is an important part of any tools used to process the specification. One simple approach is to use a representation that is easily handled directly by a particular tool e.g. $LaTeX$ source format.

$LaTeX$ is a typesetting system that is used primarily in academia. $LaTeX$ files are compatible with many academic publishers' typesetting systems. A $LaTeX$ command is a backslash '\' followed by a string of alphabetic characters, up to the first non-alphabetic characters or by a single non-alphabetic character.

Following is an example for representing a Z-Specification using $LaTeX$ Keywords: -

Z-Specification

Checkin

users, in, out : staff
name? : staff

---

name? ∈ out
in' = in ∪{name?}
out' = out \ {name?}
users' = users

Z-Specification Schema Declaration using L^AT_EX Keywords:

users, in, out : staff

name? : staff

Z-Specification Schema Predicate using L^AT_EX Keywords:

name? \in out

in' = in \cup {name?}

out' = out \ {name?}

users' = users

3.5 Problem of Searching Components

User working in Windows environment is interested in finding source code e.g. Java components to make insertion to a linked list. To accomplish this, user opens the search engine and queries the component servers providing components in the same domain in which user is interested. To query the related component servers user will use the system ontology and has to select a term from each of the facets *domain*, *type of component* and *region*. This knowledge is critical to search. A user selected *data structures* from *domain* facet, *source code* from *type of component* facet and *Internet* from *region* facet then the query will be converted to a SQL statement like *"select all from Registry-table where domain=data structures, typeofcomponent=source code, region=I."* and this will be sent for processing.

All this procedure is repeated on various servers if the user is not satisfied with the results. So searching for even a simpler component requires so much of time and manual work.

3.6 Proposed Model

Instead of performing an individual and manual search, we propose to delegate this task to mobile agents as shown in Figure 3.1. The need of searching a component can arise at two stages of the software development process, at design time and at run time. A mobile agent after searching multiple servers will bring the desired software. But, instead of searching manually through the DCR, this task is performed automatically by a mobile agent. It carries out the requirement. Once the mobile agent finishes its search [2] and presents its results, the developer will decide which component will be used in the architecture by inspecting the information about the applicants retrieved by mobile agent.

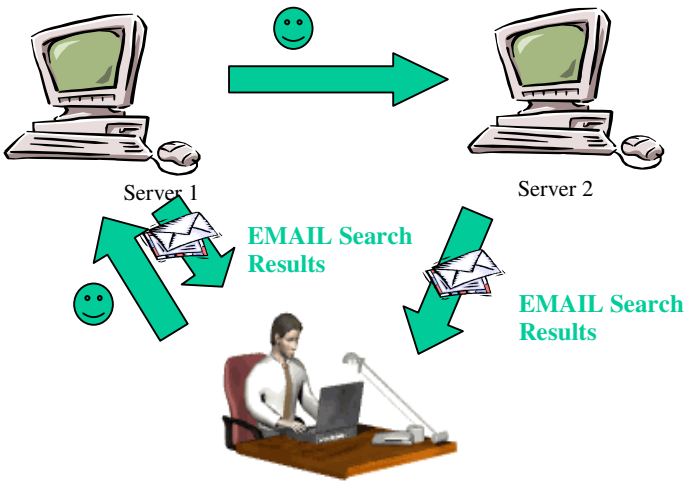


Fig.3.1: Proposed Model

The proposed model has been implemented using ASDK (Aglets Software Development Kit) having version Aglets1.1b3 [4,6,7,29] provided by IBM for developing mobile agents, JDK 1.1.8 for developing client interface and processing and JavaMail API (javamail-1.3 & jaf-1.0.2) for sending Email.

Natural and formal specifications are used for query matching. The Mobile agent after receiving the information from the user is fired from the client machine on to a server. Mobile agent carries user query to various servers for component matching. By following the search procedure, the mobile agent searches the components. After successfully executing the search operation on one server, the mobile agent submits all the results to user via email. The mobile agent then jumps to some other server and repeats the search task.

3.7 Search Procedure

In order to search a component on the Internet, the user enters the following information before sending the agent to other server: -

Component Details: -

1. Name of the Component to search.
2. Name of the function to search.

Component Description using Z-Specifications & L^at_ex Keywords: -

1. State Schema *Declaration*.
2. State Schema *Predicate* (i.e. Data Invariant).
3. Operation Schema *Precondition*.
4. Operation Schema *Declaration*.
5. Operation Schema *Predicate*.
6. Operation Schema *Post Condition*.

After entering the information described above, user presses the **Search Button**.

Query entered by the user is used first to search the components on Keyword based search [2] in the database of the server. The short listed components are further examined using formal methods, by matching the entered specifications with the stored ones.

Percentage match for each short listed component is calculated. The whole procedure for searching is explained in Figure 3.3.

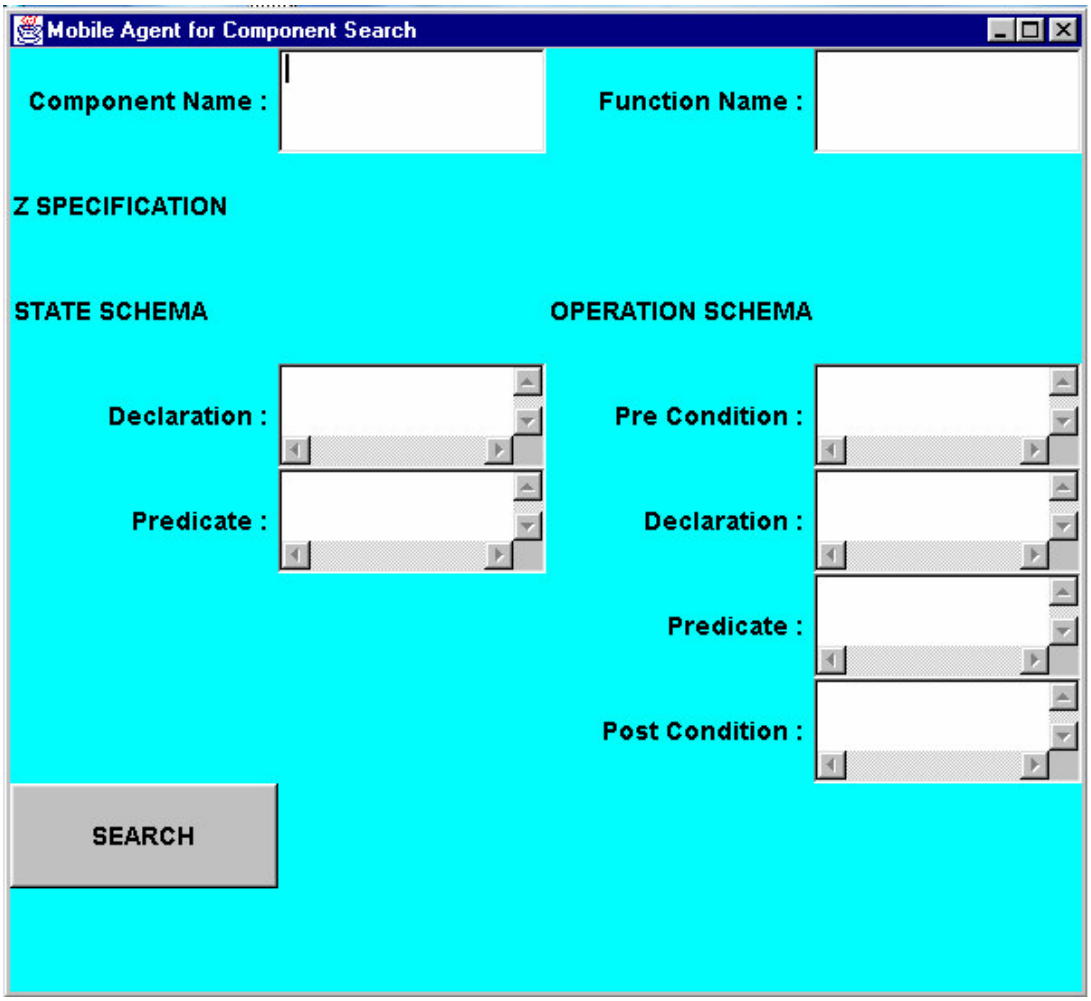


Fig. 3.2: Client Interface

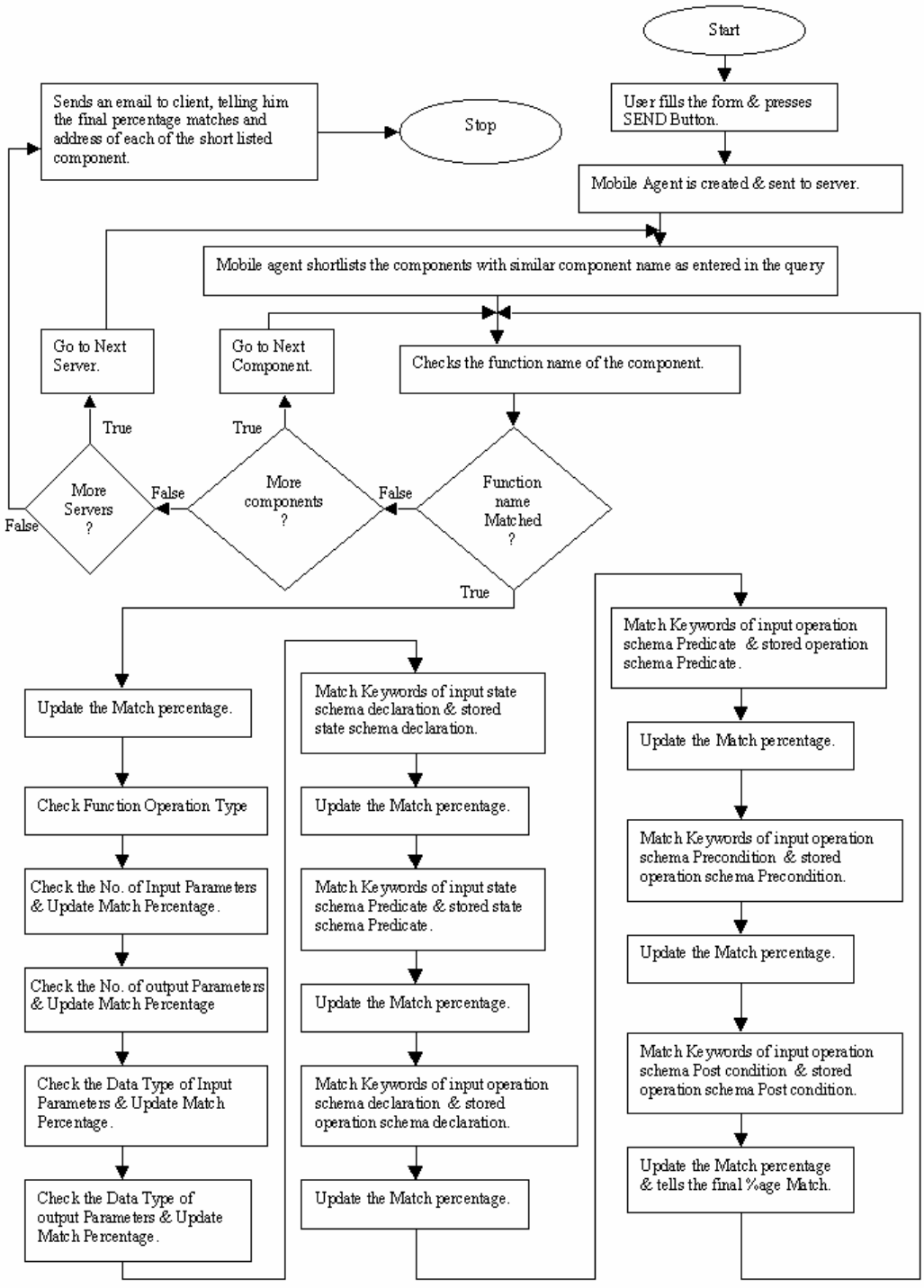


Fig. 3.3: Search Procedure

3.8 Output

The result is shown as percentage match for each component short-listed by Keyword Based Search. Following Percentage Match Equation is used for calculating the Percentage match of all the short listed components: -

$$Percentage\ Match = W_1 + a*[W_2 + W_3 + b*W_4 + c*W_5 + b*\ W_6*(N_1/NS_1+N_1/NQ_1) + c*W_7*(N_2/NS_2+N_2/NQ_2) + W_8*(N_3/NS_3+N_3/NQ_3) + W_9*(N_4/NS_4+N_4/NQ_4) + W_{10}*(N_5/NS_5+N_5/NQ_5) + W_{11}*(N_6/NS_6+N_6/NQ_6) + W_{12}*(N_7/NS_7+N_7/NQ_7) + W_{13}*(N_8/NS_8+N_8/NQ_8)]$$

Variable Name	Description
a	A variable whose value is 1 if Function Name matches else 0.
b	A variable whose value is 1 if number of input variables matches else 0.
c	A variable whose value is 1 if number of output variables matches else 0.
W_1	Weight assigned to Component Name Match.
W_2	Weight assigned to Function Name Match.
W_3	Weight assigned to Function operation Type Match.
W_4	Weight assigned to number of input variables matched.
W_5	Weight assigned to number of output variables matched.
W_6	Weight assigned to data type of input variables matched.
N_1	Number of input variables whose data types matched.
NS_1	Number of input variables in the Stored schema.
NQ_1	Number of input variables in the Query schema.
W_7	Weight assigned to data type of output variables matched.
N_2	Number of output variables whose data types matched.
NS_2	Number of output variables in the Stored schema.
NQ_2	Number of output variables in the Query schema

Table 3.1: Description of Variables

Variable Name	Description
W_8	Weight assigned to State Schema Declaration.
N_3	Number of keywords matched in the State Schema Declaration
NS_3	Number of keywords in the Stored State Schema Declaration.
NQ_3	Number of keywords in the Query State Schema Declaration.
W_9	Weight assigned to State Schema Predicate.
N_4	Number of keywords matched in the State Schema Predicate.
NS_4	Number of keywords in the Stored State Schema Predicate.
NQ_4	Number of keywords in the Query State Schema Predicate.
W_{10}	Weight assigned to Operation Schema Declaration.
N_5	Number of keywords matched in the Operation Schema Declaration
NS_5	Number of keywords in the Stored Operation Schema Declaration.
NQ_5	Number of keywords in the Query Operation Schema Declaration.
W_{11}	Weight assigned to Operation Schema Predicate.
N_6	Number of keywords matched in the Operation Schema Predicate.
NS_6	Number of keywords in the Stored Operation Schema Predicate.
NQ_6	Number of keywords in the Query Operation Schema Predicate.
W_{12}	Weight assigned to Operation Schema Precondition.
N_7	Number of keywords matched in the Operation Schema Precondition.
NS_7	Number of keywords in the Stored Operation Schema Precondition.
NQ_7	Number of keywords in the Query Operation Schema Precondition.
W_{13}	Weight assigned to Operation Schema Post condition.
N_8	Number of keywords matched in the Operation Schema Post condition.
NS_8	Number of keywords in the Stored Operation Schema Post condition.
NQ_8	Number of keywords in the Query Operation Schema Post condition.

Table 3.1(Continued): Description of Variables

Following values of weights has been taken in this implementation: -

W_1=4.75, W_2=4.75, W_3=3.8, W_4=3.8, W_5=3.8, W_6=3.8, W_7 = 3.8, W_8=7.125, W_9=7.125, W_{10}=9.5, W_{11}=9.5, W_{12}=7.125, W_{13}=7.125.

Weights are taken according to their importance in specification. These weights are so assigned that maximum percentage match should come out to be 95%, because 100% component match is not possible. Various other factors e.g. Platform, Implementation Language, Precision etc. are considered in the remaining 5%.

3.9 Case Study

Search a Component with name *attendance* having function for c*heckin*. Using component name *attendance* and function name *checkin*, a query using $L^{A}T_{E}X$ keywords is entered as shown in Figure 4.1 and results obtained are as shown in Figure 4.2. The specifications required by user are given in Table 4.1 and candidate specifications in repository are shown in Tables 4.2, 4.3, 4.4 and 4.5.

Component Name	Attendance
Function Name	Checkin
State Schema Declaration	users, in, out : staff ;
State Schema Predicate	in \cap out = { } in \cup out = users ;
Operation Schema Precondition	in \cap out = { } in \cup out = users ;
Operation Schema Declaration	\delta name? : staff age! : age ;
Operation Schema Predicate	name? \in out in' = in \cup {name?} out' = out \ {name?} users' = users ;
Operation Schema Postcondition	in \cap out = { } in \cup out = users ;

Table 4.1: Component Specifications to search

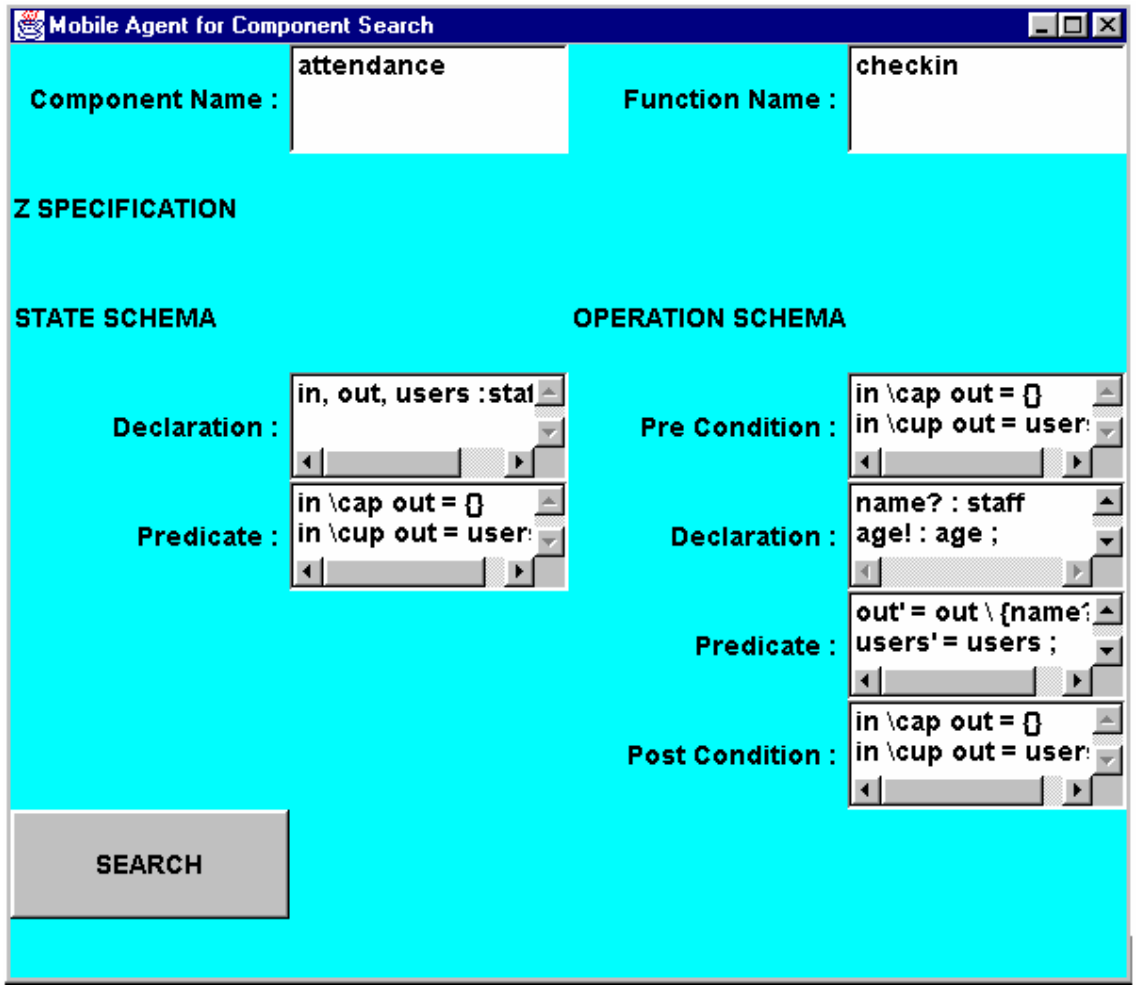


Fig. 4.1: Query

3.9.1 Specifications Stored in Repository

Consider the following specifications with component name *attendance* and function name *checkin*

Component Name	attendance
Function Name	checkin
State Schema Declaration	users, in, out : staff ;
State Schema Predicate	in \cap out = { } in \cup out = users ;
Operation Schema Precondition	in \cap out = { } in \cup out = users ;
Operation Schema Declaration	\delta name? : staff age! : age ;
Operation Schema Predicate	name? \in out in' = in \cup {name?} out' = out \ {name?} users' = users ;
Operation Schema Postcondition	in \cap out = { } in \cup out = users ;

Table 4.2: Component1 Specifications

Component Name	attendance
Function Name	checkout
State Schema Declaration	users, in, out : staff ;
State Schema Predicate	in \cap out = { } in \cup out = users ;
Operation Schema Precondition	in \cap out = { } in \cup out = users ;
Operation Schema Declaration	\delta name? : staff no! : no age! : staff ;
Operation Schema Predicate	name? \in out in' = in \cup {name?} out' = out \ {name?} users' = users ;
Operation Schema Postcondition	in \cap out = { } in \cup out = users ;

Table 4.3: Component2 Specifications

Component Name	attendance
Function Name	checkin
State Schema Declaration	users, in, out : staff ;
State Schema Predicate	in \cap out = { } in \cup out = users ;
Operation Schema Precondition	in \cap out = { } in \cup out = users ;
Operation Schema Declaration	\delta name? : staff no! : no age? : staff ;
Operation Schema Predicate	name? \in out in' = in \cup {name?} out' = out \ {name?} users' = users ;
Operation Schema Postcondition	in \cap out = { } in \cup out = users ;

Table 4.4: Component3 Specifications

Component Name	attendance
Function Name	checkinout
State Schema Declaration	users, in, out : staff ;
State Schema Predicate	in \cap out = { } in \cup out = users ;
Operation Schema Precondition	in \cap out = { } in \cup out = users ;
Operation Schema Declaration	\delta name? : staff abc? : abc no! : no age? : staff ;
Operation Schema Predicate	name? \in out in' = in \cup {name?} out' = out \ {name?} users' = users ;
Operation Schema Postcondition	in \cap out = { } in \cup out = users ;

Table 4.5: Component4 Specifications

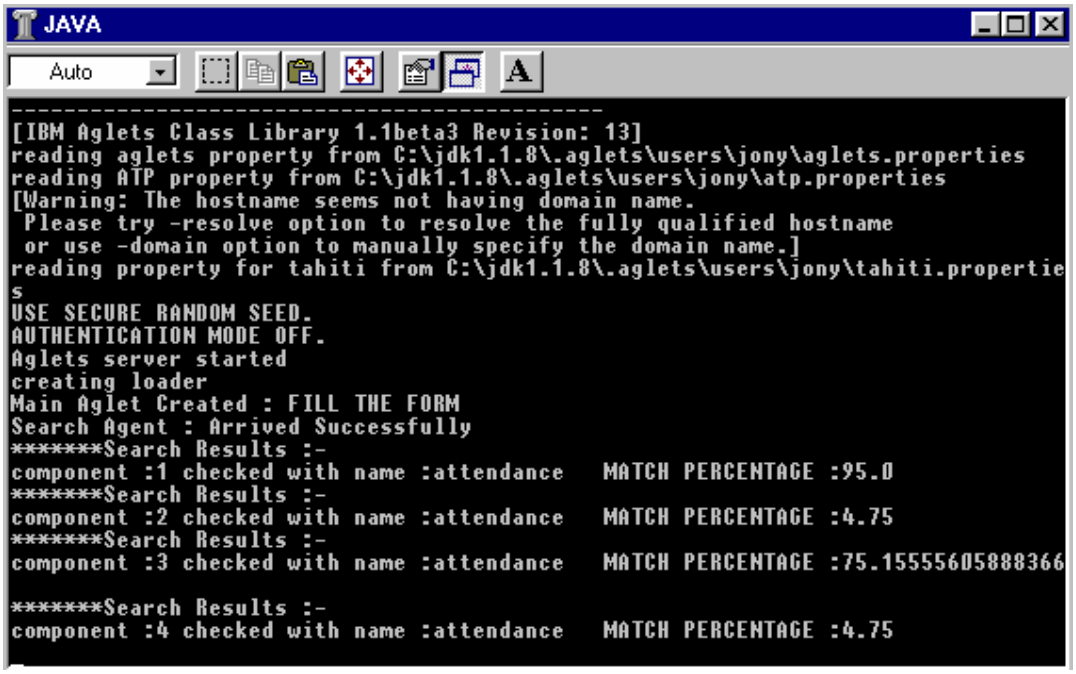


Fig 4.2: Search Results

Percentage match for each short listed component is calculated. The whole procedure for searching is explained below: -

1. User enters the required specifications and presses **Search Button**.
2. Mobile agent is created and sent to the server.
3. After reaching the server, Mobile Agent shortlists the components having *Component Name* matching with the entered *Component Name i.e. "attendance"*.

Four components are short listed as shown previously.

Calculations for Component1 with specifications shown in Table 4.2: -

I. For Component1 Percentage Match=4.75

II. Match *Function Name* of the *component1* i.e. *"checkin"* and the *Function name* in the Query i.e. "checkin". Function name matched.

Percentage Match=4.75+4.75=9.5

III. Match Function Operation Type

Percentage Match=9.5+3.8=13.3

IV. Match number of input variables to the function and calculates the *Percentage Match* for the Component.

Percentage Match=13.3+3.8=17.1

V. Match number of output variables to the function and updates the *Percentage Match* for the Component.

Percentage Match=17.1+3.8=20.9

VI. Match Data type of input variables to the function and updates the *Percentage Match* for the Component.

Percentage Match=20.9+3.8=24.7

VII. Match Data type of output variables to the function and updates the *Percentage Match* for the Component.

Percentage Match=24.7+3.8=28.5

VIII. Match Keywords in *State Schema Declaration* and updates the *Percentage Match* for the Component.

Percentage Match=28.5+7.125=35.625

IX. Match Keywords in *State Schema Predicate* and updates the *Percentage Match* for the Component.

Percentage Match=35.625+7.125=42.75

X. Match Keywords in *Operation Schema Declaration* and updates the *Percentage Match* for the Component.

Percentage Match=42.75+19.0=61.75

XI. Match Keywords in *Operation Schema Predicate* and updates the *Percentage Match* for the Component.

Percentage Match=61.75+19.0=80.75

XII. Match Keywords in *Operation Schema Precondition* and updates the *Percentage Match* for the Component.

Percentage Match=80.75+7.125=87.875

XIII. Match Keywords in *Operation Schema Post condition* and updates the *Percentage Match* for the Component.

Percentage Match=87.875+7.125=95.0

Hence percentage match for the Component1 comes out to be 95%.

Calculations for Component2 with specifications shown in Table 4.3: -

I. For Component2 Percentage Match=4.75

II. Match *Function Name* of the *component2* i.e. "*checkout*" and the *Function name* in the Query i.e. "checkin". Function name not matched.

Percentage Match=4.75

Hence percentage match for the Component2 comes out to be 4.75%.

Calculations for Component3 with specifications shown in Table 4.4: -

I. For Component3 Percentage Match=4.75

II. Match *Function Name* of the *component3* i.e. *"checkin"* and the *Function name* in the Query i.e. "checkin". Function name matched.

Percentage Match=4.75+4.75=9.5

III. Match Function Operation Type

Percentage Match=9.5+3.8=13.3

IV. Match number of input variables to the function and calculates the *Percentage Match* for the Component.

Percentage Match=13.3+0=13.3

V. Match number of output variables to the function and updates the *Percentage Match* for the Component.

Percentage Match=13.3+3.8=17.1

VI. Match Data type of input variables to the function and updates the *Percentage Match* for the Component.

Percentage Match=17.1+0=17.1

VII. Match Data type of output variables to the function and updates the *Percentage Match* for the Component.

Percentage Match=17.1+0=17.1

VIII. Match Keywords in *State Schema Declaration* and updates the *Percentage Match* for the Component.

Percentage Match=17.1+7.125=24.225

IX. Match Keywords in *State Schema Predicate* and updates the *Percentage Match* for the Component.

Percentage Match=24.225+7.125=31.35

X. Match Keywords in *Operation Schema Declaration* and updates the *Percentage Match* for the Component.

Percentage Match=31.35+10.56=41.9

XI. Match Keywords in *Operation Schema Predicate* and updates the *Percentage Match* for the Component.

Percentage Match=41.9+19.0=60.9

XII. Match Keywords in *Operation Schema Precondition* and updates the *Percentage Match* for the Component.

Percentage Match=60.9+7.125=68.0

XIII. Match Keywords in *Operation Schema Post condition* and updates the *Percentage Match* for the Component.

Percentage Match=68.0+7.125=75.16

Hence percentage match for the Component3 comes out to be 75.16%.

Calculations for Component4 with specifications shown in Table 4.5: -

I. For Component4 Percentage Match=4.75

II. Match *Function Name* of the *component4* i.e. "*checkinout*" and the *Function name* in the Query i.e. "checkin". Function name not matched.

Percentage Match=4.75

Hence percentage match for the Component4 comes out to be 4.75%.

4. After the whole calculations, mobile agent sends an email to the client telling the final result i.e. Percentage Match of each component.

3.10 Benefits

1. Results are shown on the basis of percentage match, helps us to figure out how much the component needs modification for its reuse.
2. Since mobile agents do not need persistent connection, it is useful for dialup users to search a component on web and also saves the bandwidth.

3. It works autonomously, switches from one server to another without user interaction.
4. Hybrid approach for searching a component is used, thus providing more useful results.
5. In single run this system is able to go to a number of servers without user notification and perform the search operations. This decreases user's task of repetitive search on different search engines and repositories.
6. User does not need to wait online at the time of query processing, so it saves users time as well as bandwidth. The user can check the status of the query later on.

In a modified form of this system, user can receive an email about its query results rather than checking second time on the server and hence can reduce the user's effort to search a component on web.

Chapter 4
Conclusions & Future Scope

This model is implemented using keyword and formal specifications. Keyword matching is used at higher level to match component name and function name. Formal methods are further used at schema and predicate level. Formal specifications are used for appropriate component matching. A formal specification helps to search most appropriate component.

4.1 Conclusions

The model proposed in current work provides a mobile agent based software retrieval service that provides software developers with a mechanism to search and retrieve the required reusable component from various repositories at different locations i.e. distributed repositories. Although this service can be used for fixed computers it puts special emphasis on optimizing the use of battery and wireless communications, as user can disconnect the network connection after sending the mobile agent and reconnect later to check results. Some important results are shown below: -

- Proposed model is capable of searching a component with percentage match ranging from a minimum of 4.75% to a maximum of 95%.
- A minimum match occurs when only component name matches as component name has some contribution in the overall percentage match.
- A maximum match occurs when an entered specification exactly matches the stored specifications.

4.2 Future Scope

A major concern specific to mobile agents is the protection of the servers running the agents. Running arbitrary programs on a machine is dangerous as a hostile program could destroy the hard drive, steal data, or do all sorts of other undesirable things. This risk must be thoroughly addressed if mobile agent environments are to succeed.

The complement of server security is agent security: whether the agent can trust the server on which it is executing. A mobile agent might contain secret information such as proprietary data and algorithms. Servers can steal the secret information.

A lot of research is still pending to make use of mobile agents secure. The area of providing security is still in developing stage. A server needs protection against malicious agents that can harm the machine.

Various other possible design patterns for the proposed model are as follows: -

- Mobile agent can reply back to client after visiting all the servers as shown in Fig. 4.3.

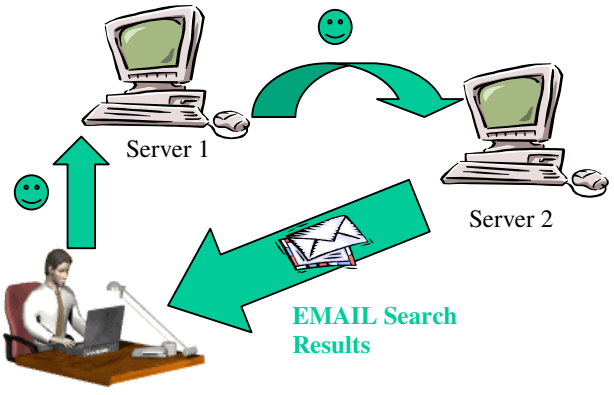


Fig. 4.3: Design Pattern I

- Mobile agent can jump back to client after visiting one server at a time as shown in Fig. 4.4.

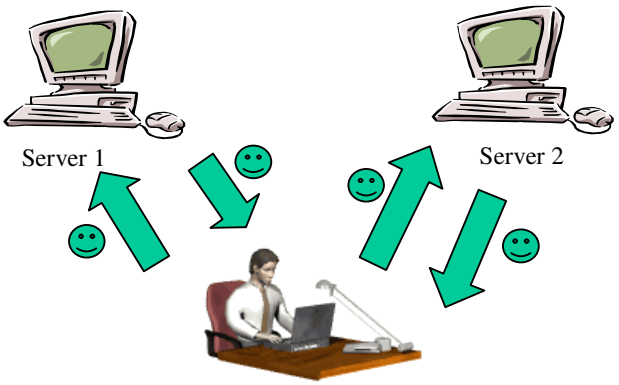


Fig. 4.4: Design Pattern II

- Mobile agent can submit results on a central server and user can check the result from server as shown in Fig. 4.5. The storage of queries and replies to the server will help developers in future component searches.

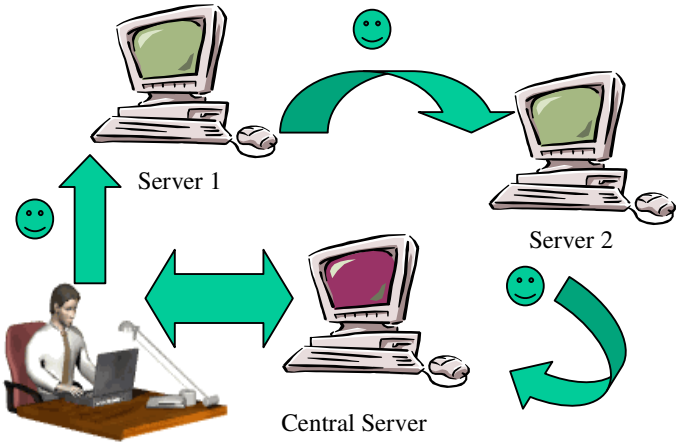


Fig. 4.5: Design Pattern III

Appendix - I

Glossary of Z Notation

Names

a,b	**identifiers**
d,e	**declarations (e.g.,** a: A; b, ...: B ...**)**
f,g	**functions**
m,n	**numbers**
p,q	**predicates**
s,t	**sequences**
x,y	**expressions**
A,B	**sets**
C,D	**bags**
Q,R	**relations**
S,T	**schemas**
X	**schema text (e.g.,** d, d | p**, or** S**)**

Definitions

a == x	**Abbreviation definition**
a ::= b | ...	**Free type definition**
[a]	**Introduction of a given set (or** [a,...]**)**
a _	**Prefix operator**
_ a	**Postfix operator**
_ a _	**Infix operator**

Logic

true	**Logical true constant**
false	**Logical false constant**
¬p	**Logical negation, *not***
p ∧q	**Logical conjunction, *and***
p ∨q	**Logical disjunction, *or***
p ⇒q	**Logical implication**
p ⇔q	**Logical equivalence**
∀X •q	**Universal quantification**
∃X •q	**Existential quantification**
(**let** a == x; ... •p)	**Local definition**

Sets and expressions

x = y	**Equality**
x ≠y	**Inequality**
x ∈A	**Set membership**

x ∉A	**Non-membership**
∅	**Empty set**
A ⊆B	**Set inclusion**
A ⊂B	**Strict set inclusion**
{ x, y, ... }	**Set display**
{ X •x }	**Set comprehension**
(λ X •x)	**Lambda expression**
(**let** a == x; ... •y)	**Local definition**
if p **then** x **else** y	**Conditional expression**
(x,y, ...)	**Tuple**
(x,y)	**Pair**
A ×B ×...	**Cartesian product**
ℙA	**Power set**
A ∩B	**Set intersection**
A ∪B	**Set union**
A \ B	**Set difference**
first x	**First element of an ordered pair**
second x	**Second element of an ordered pair**
# A	**Number of elements in a set**

Relations

A ↔B	**Binary relation (**ℙ (A ×B)**)**
a ↦b	**Maplet (**(a,b)**)**
dom R	**Domain of a relation**
ran R	**Range of a relation**
Q ⨾R	**Forward relational composition**
Q ∘R	**Backward relational composition (**R ⨾Q**)**
A ◁R	**Domain restriction**
A ⩤R	**Domain anti-restriction**
A ▷R	**Range restriction**
A ⩥R	**Range anti-restriction**
R (|A |)	**Relational image**
R ~	**Inverse of relation**
R +	**Transitive closure**
Q ⩤R	**Relational overriding**
a R b	**Infix relation**

Functions

A ⇸B	**Partial functions**
A →B	**Total functions**
A ⤔B	**Partial injections**
A ↣B	**Total injections**
A ⤖B	**Bijections**
f x	**Function application (or** f(x)**)**

Numbers

$\mathbb{Z}$	**Set of integers**
$\mathbb{N}$	**Set of natural numbers** { 0, 1, 2, ... }
$\mathbb{N}_1$	**Set of strictly positive numbers** { 1, 2, ... }
m+n	**Addition**
m-n	**Subtraction**
m*n	**Multiplication**
m **div** n	**Division**
m **mod** n	**Remainder (modulus)**
m ≤n	**Less than or equal**
m < n	**Less than**
m ≥n	**Greater than or equal**
m > n	**Greater than**
m .. n	**Number range**
min A	**Minimum of a set of numbers**
max A	**Maximum of a set of numbers**

Sequences

seq A	**Set of finite sequences**
seq_1 A	**Set of non-empty finite sequences**
iseq A	**Set of finite injective sequences**
$\langle\rangle$	**Empty sequence**
⟨x, y, ... ⟩	**Sequence display**
s ⌢t	**Sequence concatenation**
head s	**First element of a sequence**
tail s	**All but the head element of a sequence**
last s	**Last element of a sequence**
front s	**All but the last element of a sequence**
s **in** t	**Sequence segment relation**

Schema Calculus

S ≙[X]	**Horizontal schema**
[T; ... | ...]	**Schema inclusion**
z.a	**Component selection (given** z:S**)**
θS	**Binding**
¬S	**Schema negation**
S ∧T	**Schema conjunction**
S ∨T	**Schema disjunction**
S ⨾T	**Schema composition**
S >> T	**Schema piping**

Conventions

a?	**Input to an operation**
a!	**Output from an operation**
a	**State component before an operation**
a’	**State component after an operation**
S	**State schema before an operation**
S’	**State schema after an operation**
ΔS	**Change of state**
ΞS	**No change of state**

References

[1] Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna, "Understanding Code Mobility", IEEE Transactions on the Software Engineering, Volume 24, No. 5, May 1998.

[2] Amy Moormann Zaremski, Jeannette M. Wing,"Specification Matching of Software Components", ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 4, Pages 333-369, October 1997.

[3] Bill McCarty and Luke Cassady-Dorion , "Java Distributed Objects" , Techmedia, 1999.

[4] Damir Horvat, Dragana Cvetkovie, Veljko Milutinovie, Peter Kocovic and Vlada Kovacevie . "Mobile Agents and Java Mobile Agents Toolkits"

[5] Danny B. Lange, "Mobile Objects and Mobile Agents: The Future of distributed Computing"

[6] Danny B. Lange, Mitsuru Oshima, "Mobile Agents with java : The Aglet API".

[7] D.B Lange and Mitsuru Oshima, "Programming and Deploying Java Mobile Agents with Aglets", Addison-Wesley, 1998.

[8] David Chess, Colin Harrison, Aaron Kershenbaum, "IBM Research Report".

[9] David Kotz, Robert S. Gray, "Mobile Agents and the Future of the Internet", ACM Operating Systems Review 33(3), pp. 7-13, August 1999.

[10] David Kotz, Robert Gray, and Daniela Rus , http://dsonline.computer.org/0208/f/kot_print.htm

[11] Dr. DONG Jin Song, "Formal Specification and Design Techniques".

[12] Girardi Gutierrez," Classification and retrieval of software through their Descriptions in Natural Language".

[13] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile Agent Technology : Current Trends and Perspectives".

[14] GMD FOKUS IBM, "Mobile Agent System Interoperability Facilities Specification", OMG TC Document orbos/97-10-05, November 10, 1997.

[15] Ivar Jacobson, Martin Griss and Patrik Jonsson, "Software Reuse Architecture, Process and Organisation for Business Success", Addison Wesley, 2000.

[16] Jan Vitek, "New Paradigms for Distributed Programming", Proceedings European Research Seminar in Advanced Distributed Systems (ERSADS'97), March 17-21, 1997.

[17] Jun-Jaang Jeng , Betty H.C Cheng, "Using Formal Methods to Construct a Software Component Library", Proceedings of 4th Software Engineering Conference, Lecture Notes in Computer Science, Vol 717, pp. 397-417, September 1993.

[18] Kenji Sugawara,"An Agent-based Framework for Developing Flexible Distributed Systems", Proceedings of First IEEE International Conference on Cognitive Informatics (ICCI), 2002.

[19] Kyriakos Koukoumpetsos and Nokos Antonopoulos, "Mobility Patterns: An Alternative Approach to Mobility Management"

[20] L. Ismail, D. Hagimont, "A Performance Evaluation of the Mobile Agent Paradigm".

[21] M.Amor, M.Pinto, L. Fuentes and J.M.Troya, "Combining Software Components and Mobile Agents".

[22] Paulo Marques, Paulo Simoes, Luis Silva, Fernando Boavida, Joa Silva, "Providing Applications with Mobile Agent Technology", IEEE OPENARCH, 2001.

[23] Reinhard Riel, Takashi Suezawa, "Management of Information Markets with Mobile Software Agents".

[24] Roger S. Pressmann, "Software Engineering- A Practitioner's Approach", McGraw-Hill, Fifth Edition.

[25] Simarjot Singh, Rajesh Kumar Bhatia, "Software Component Reuse Using Formal Methods & k-nn Technique", Proceedings of the International Conference on Software Engineering Research and Practice, SERP '03, Vol.2, June 23 – 26, 2003.

[26] Wei Quiang, Hinny Kong Pe Hin, "Agent Based System for Mobile Commerce", Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), 2002.

[27] Yunwen Ye, Gerhard Fisher,"Promoting Reuse with active Reuse Repository Systems", Proceedings of 6th International Conference On Software Reuse (ICSR-6), Springer-Veriag, Vienna, Austria, pp. 302-317, June 27-29, 2000.

Web Sites

[28] http://aglets.sourceforge.net/.

[29] http://www.trl.ibm.com/aglets/.

[30] http://mole.informatik.uni-stuttgart.de/mal/preview/

[31] http://www.agentry.net/

[32] http://www.info.uqam.ca/~mili/Enseignement/MIG8500-s00/book-outline.html

[33] http://fmt.cs.utwente.nl/

[34] http://www.afm.sbu.ac.uk/

[35] http://www.cafm.sbu.ac.uk/

[36] http://www.cs.indiana.edu/formal-methods-education/

[37] http://www.ieee.org/

[38] http://www.cs.mu.oz.au/~leon/agentlinks.html

[39] http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html

[40] http://www.ecs.soton.ac.uk/~mml/absd/resources.html

[41] http://xenia.media.mit.edu/~nelson/research/dc/node2.html

[42] http://agents.umbc.edu/

List of Papers

1. Amandeep Singh, Rajesh K. Bhatia, Dr. R. C Joshi and Dr. Mayank Dave, "Mobile Agents in Reusable Software Component Retrieval " communicated to The 2004 International Workshop on Mobile Systems, E-commerce and Agent Technology (MSEAT'2004) in conjunction with the 9th International Conference on Distributed Multimedia Systems (DMS'2004). (Communicated on 15th May 2004).
2. Amandeep Singh, Rajesh K. Bhatia, Dr. Mayank Dave and Dr. R. C Joshi, "Agent Based Innovative Approach for Component Retrieval" communicated to Computer Society of India Chapter May 2004.