Distributed Component Hub for Reusable Software Components Management

James X. Ci and Wei-Tek Tsai
Computer Science and Engineering Department, University of Minnesota,
Minneapolis, MN 55455
Email: {xci, tsai}@cs.umn.edu

Abstract

This paper presents a distributed software component hub model for reusable software components management, with LDAP operations and controls. It first briefly discusses the emerging software component technology and its major issues currently under seeking. Then we describe the methods of distributed component hub for effectively resolving the hassles in reusable software components repository management and retrieval. We specify the overall distributed component hub's framework and its design architecture. The hub's core mechanisms show how reusable software components repository and retrieval can be effectively managed with the distributed component hub framework. The experimentation demonstrates the model's promise.

***Keywords**: Componentbased Software Engineering, Software Component Retrieval, Repository Management, Software Component Reuse Environments.*

1. Introduction

Reusable software components repository and retrieval management have long been in active research for years [Boo87, Pri91, Bor95, Beh98, SHW98, Griss99]. Componentization liberates software engineers from bits and bytes to problem solving methods and problem analysis techniques. With the increasing use of distributed collaborative software development and component-based engineering, the importance of distributed software component reuse and management has been considerably addressed [Bro97,MOT97, Szy98 ,BW99]. In today's networked environments, hassles and difficulties on managing reusable software components for effective and efficient software components reuse are confronted onward at different levels[Bro97, Szy98, MMM98, BW99]: (1) Making reusable software components broadly known and available over a distributed and heterogeneous environment for different component types based on a number of different interaction standards; (2) Managing distributed software components repositories over a highly distributed environment, particularly an enterprise Intranet; and (3) Facilitating software components retrieval for effective reuse and component-based engineering.

Component-based software paradigms commonly provide five basic types of services: component and its interfaces publishing and discovery, event handling, persistence, layout control, and application builder support. Traditional approaches to software component reuse are characterized by uncertainty, complexity,and excessive efforts[Boo87,Pri91,Bor95]. Few work deals with distributed software component reuse management, particularly over a network of heterogeneous computer systems, for these basic types of services.

In the rest of the paper we describe a distributed component hub framework model, with LDAP [HS97] operations and controls, for systematically managing reusable software components. It serves as generic building blocks to facilitate the construction process of distributed software component-based applications and reuse. Experimentation work demonstrates the model's promise. Discussions on related work and a short conclusion round out the paper.

2. Distributed Component Hub

In component-based software paradigm, distributed software components are binary or open source units of independent production, self-contained with two ways plug-in/out interaction capability. There is a clear separation between software component development and component-based application development, similar to the differences between chip design and circuit design. And in between there are components management and reuse control. By software components as well as so called reusable software assets, it also means off-the-shelf software components[MOT97], public domain software and shareware. In component-based software design and development scenarios, for various reuse purposes, software components are clustered into modules, framework, use-case roles/functionalities, and sometimes with design patterns and implementation packages. [Bro97] and [Beh98] indicate that a collection of modules randomly contributed by software components producers or suppliers does not produce an effective storage of tightly-related components.

Domain dependent and application specific reuse scenarios in highly network-centric environment and framework deduce that software components ought to be clustered into categories and systematically managed as a hub for distributed access of reusable software components spread over a heterogeneous network. It is practically difficult and not cost effective to organize and store reusable software components by arbitrary domains over a randomly scattered network for general purpose. Generality and efficiency appear to be contradictory requirements. And specifically software component repository and retrieval infer that creating and using naming contexts for retrieval closely tight to the scenarios and framework used in software design and reuse process could considerably improve retrieval process on its cost and actuary. In contrast, traditional approaches on software components retrieval rely heavily on automated search mechanisms alone [MMM98, SHW98]. Domain content dependent and application specific reuse index mechanism augmented by naming server bring in efficiency and effectiveness to various retrieval operations, in addition to automated search mechanisms alone.

2.1 Reuse-Index Mechanism

Reusable software components index mechanism is considered as augment to automated software components search process and operations. It not only serves as a mean to organize reusable software components in storage but also functions as looking-up device for retrieval. Reusable components first register and publish themselves with interfaces, along with their reusable contexts, e.g. framework.

The reusable index consists of a tree structure of reusable components data, which is an information holder of abstractions of objects which are interconnected as a tree hierarchy. Its contents are meta-data of software components, obtained elsewhere. The basic unit of the tree *T* is an entry, a collection of information about an object, which is composed of a set of attributes, each describes one particular trait of the object. Every entry is treated as an object and object can be embedded within objects. Each attribute consists of a type with an associated syntax, and one or more values. Although it is logically a single tree, *T* can be distributed physically across many machines. Figure 2.1.1 shows an excerption of a typical instance of *T*.

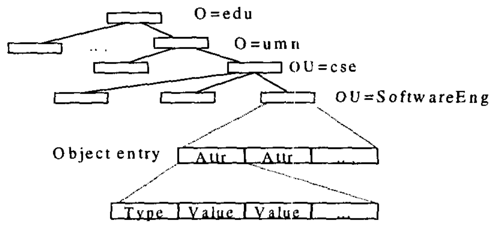


Figure 2.1.1

In a distributed collaborative software environment, software component reuse and component-based software process call for reusable components, together with their reusable contexts. The reusable index provides such citation management for their reusable contexts, which extends component publishing and discovery.

For any reusable component indexed, its reusable context is cited by its parent node and siblings. If a reusable component is an opaque, its reuse context can be described by the indexed entry attribute, which is further referred by an URL of its reuse specification. The index node itself contains all information regarding the components such as the component name, functionality description, interfaces, and component type and platform, which all are listed as the attributes of the index object. In order to keep track of the actual location of the binary files and reuse specification documents, some attributes memorize the URL address of these files, allowing a file to be physically placed anywhere in the Internet.

An alias node is a node entry in the reusable index for indirectly referring to an another node in the index. It functions as a virtual index node placeholder. By alias, reusable context citation can be flexibly extended for components with multiple reusable contexts.

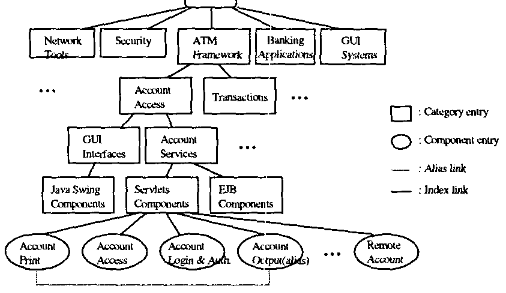


Figure 2.1.2

For indexing and classifying components in order to make searching and browsing operations efficient, the reuse index is organized as context-based categorization hierarchy of domain/domain-partitioning, functionality/function-decomposition, framework/framework-packaging and component-types/tasks breakdowns. This context-based categorization, in addition to component entry schema-based classification, reflects and projects the observation that reusable software components are reconceived and reused with domain-based contexts. Reusability across different domains with generality or isolations is deemed very rare. Figure 2.1.2 shows an excerpt of the domain-based categorization hierarchy of index, being used in the experimentation of the approach work of this paper.

2.2 Reusable Components Naming Server and Lookup

The Reusable components naming server *N* defines the naming context of the index tree ***T***, that is, the organization and reference of entry names. The name of an index entry is formed by connecting in a series all the individual names of the parent entries back to the root. For instance, in Figure 2.1.1 the rightmost entry's name from the bottom is *ou= SoftwareEng, ou= cse, o = umn, o = edu.* The individual components of the entry's name are separated by commons. This naming context gives a unique name to any entry in the hub model, that is, this distinguished name (DN) is how any entry in ***T*** is referred. In any entry's DN, the leftmost component is called the relative distinguished name (RDN). However, each RDN must be unique among its peers.

With the index naming instrument of the hub model, any indexed and published components entries have an unique name and identification within the distributed component hub. The advantage of using such naming instrument is to considerably enhance the effectiveness of repository management and retrieval efficiency. The users of the reuse-index are able to narrow the search space to specific domain or context category by cutting off the index tree through the index naming instrument. The context-based categorization hierarchy along with the index naming server helps create components storage being able to retrieve more easily, leading to a de-compositional approach guided by the naming path and the index hierarchy.

This index naming instrument can be fully consistent with URL convention. Once a particular reusable component is retrieved, its unique name (DN) and context can be memorized and stored for later reuse reference, in comparison with the traditional retrieval approach that each retrieval request has to be conducted by search mechanisms.

The following is a lookup class in Java for a specified set of reusable components upon the reuse-index via naming server.

```
//Reuse-index lookup by naming context

public class Lookup
{
//initial context implementation
public static string  LOOKUPBASE = "o=umn.edu";
public static string LOOKUPCONTEXT =
"edu.umn.cs.selab.hubEngine.testComponentStore";
public static string RETRIEVE_FILER =
"(cn=Tester)";

public static void main(String args[])
{
  try
    {
    //Hashtable for category data
    Hashtable  index = new Hashtable();
   //Specifying Index class to use
index.put(Context.Initial_Context,
                         LOOKUPCONTEXT);
  //Get a reference to an Index context
  IndexContext  ctx = new
                InitialIndexContext(index);
//Specify the scope of the lookup
LookupControls constraints = new LookupControls();
constraints.setLookupScope(
          LookupControls.SUBTREE_SCOPE);
//Perform the lookup
NamingEnumeration results =
           ctx.lookup(MY_LOOKUPBASE, MY_FILTER,
                        constrains);
//Now step through the lookup results
while (results !=null && results.hasMore())
{
//get the results
}
catch(Exception e)
{
  e.printStackTrace();
system.exit(1);
}
}
}
```

2.3 Distributed Component Hub Scheme

In order to effectively support distributed collaborative component-based software reuse, reuse-index and naming instrument are augmented with distributed component hub scheme and interoperable access mechanisms.

- ***Uniform Representation of Components:*** All software component artifacts are classified, categorized and stored by using a uniform object-based representation. And components can be easily identified and accessed via naming instrument. This uniform abstraction of physical software components in a distributed repository provides a common accessible interface to all users, applications as well as other components tools. It simplifies and supports similar software components comparison and index hierarchy navigation. Component interfaces are specified in IDL.

- ***Dynamic Context-based Categorization Hierarchy:*** As component-based tools and application domains keep fast changing, the index hierarchy is in open structure in order to support dynamic changes and design-for-change and design-for-reuse notions. This open hierarchy is implemented by LDAP schema operations, as discussed in a later section.

- ***Framework for Distributed Component Repositories Integration:*** The software components stored in distributed repositories are indexed and accessed through a hub, the framework for integrating various distributed software components storage, repositories. This integration helps in harnessing the power and functionality of the various existing component reuse tools to generate new value-added functionality than individual tools.

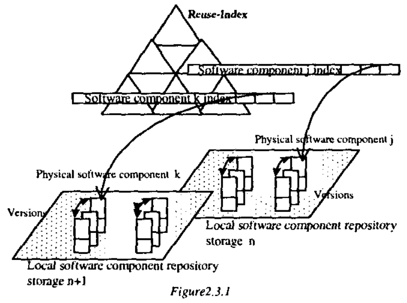

Reuse-Index
Software component j index
Software component k index
Physical software component j
Physical software component k
Versions
Versions
Local software component repository storage n
Local software component repository storage n+1

Figure2.3.1

Figure 2.3.1 shows the structure and connection of the reuse index and the distributed repositories of software components.

3. Hub Architecture and Organization

For proof-concept and assessment of the research approach, we have devised and implemented a directory-based components hub prototype for software component management and retrieval, in *Netscape*'s enterprise LDAP server, with XML [Bra+98] support by Microsoft Internet Explorer, upon a mixture of Unix and Microsoft NT networked environment.

Establishing a LDAP-based software component hub model with XML involves a number of tasks. These tasks include specifying the overall hub architecture, designing index schema and hierarchical tree of entries, organizing and indexing software components in proper categories for effective and efficient retrieval, specifying the abstraction of contents of a collection of software component descriptions, and defining software component attribute types and values.

3.1 Prototype Architecture

A three-tier architecture is used for the distributed software component hub model. The model consists of following modules and interoperations among the three tiers:

- The first tier is designed for user interface logic and is Java-enabled and web-based in XML/HTML. The major functions for the first tier's modules are client access and hub administration.

There are three kinds of clients, (1)reusable component consumer/end-user, (2) software component producer/supplier, and (3) hub administrator. This first tier is the hub's front-end responsible for the interfaces of reusable component search, look-up, index hierarchy browsing/navigation, and software component check-in, check-out, and retrieval.

- The second tier is specified for software component index and retrieval management. This middle tier is the core responsible for managing software components in proper categorization with LDAP operations and controls.

It processes requests from first tier and manages control flow back and forth between web server and LDAP server. A software component index DIT(Directory Information Tree)[HS97] is built and dynamically maintained in this middle tire, with a hierarchical structure, schema of objects with attribute types and values for reusable component abstraction.

- The third tier is the back-end of the hub model responsible for software component repository management particularly in a highly distributed manner, with distributed software component version control.

This back-end tier is devised and implemented by HTTP/FTP server with file systems under Unix and NT environments. HTTP/FTP are able to support upload and download physical reusable components. Z39.50[AN95] can also be considered as the communication protocol for the repository access.

The front-end tier communicates with the middle tier via HTTP, XML and LDAP protocols. We use XML both as the web interface tool and the communication protocol between the tiers for loading software component search results. Java Applets are used by administrator client for the indirect LDAP-based reuse-index and naming server access and components repository access. Java Servlets are used for reuse-index lookup and search mechanisms such as search on software component ID, name, type, and subject. The servlets convert LDAP search results to XML and carry on components repository operations as well.

3.2 Reuse-Index Organization

The initial hierarchy of the reuse-index tree is imposed through LDIF (LDAP Data Interchange Format)[HS97]. We also use LDIF for importing and exporting entries into and from the prototype. The following is a typical software component reuse-index entry represented in LDIF:

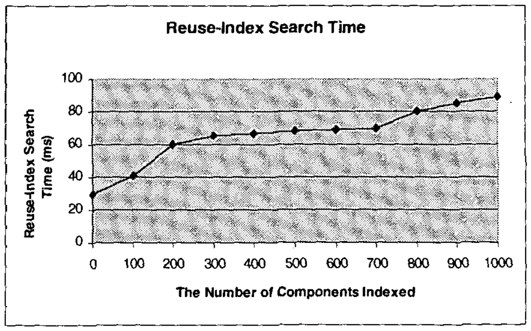
dn: id=DateComponent, dc=Transaction, dc=ATMProject, dc=Root
objectClass: top
objectClass: softwareComponent
ComponentName: CurrentDate
ComponentURL:
ftp://selab1.cs.umn.edu/hubengine/repository/currentDate
ComponentType: Java
FunctionDescription: To output the current date

The LDIF specification is also used for dynamic check-in of software components. Due to the loose coupling between reuse-index and physical software component, the hierarchy of reuse-index can be dynamically restructured without affecting software components repository.

4. Approach Assessment

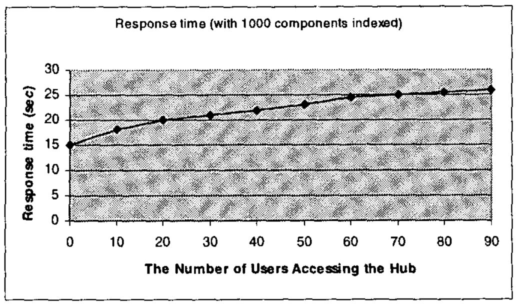
As LDAP operations and control provide strong generic and comprehensive classification and indexing capabilities, we tried out different ways for reusable components categorization for retrieval such as the faceted classification[Pri91], Kain's categorization[Kai96], Dusink and Katwijk categorization[DK87] and Wegner's software component classification[Weg89]. Each of those categorizations has its own merits. Among them, the distributed component hub approach presented in this paper demonstrates the promise and potential. The experimentation shows that the LDAP-based distributed component hub model is a feasible, effective and efficient solution for reusable software component management. It can accommodate Java components, ORB components and COM components. The former includes Beans or RMI objects which must be categorized into class paths to avoid naming conflicts. The latter, COM components rely on their GUID to differentiate themselves instead of forming a class hierarchy.

The first chart below shows the Reuse-Index performance results of a search request with the same fixed filter on *cn*. The performance was measured on a workstation of 64Mb RAM, 200Mhz CPU running



Microsoft Window NT4.0. The Reuse-Index is implemented in *Netscape*'s LDAP server 4.0 under indexed mode.

The second chart below illustrates the distributed components hub's performance results of a search request with filter on *cn*. The performance was measured in the same environment as aforementioned, while the Internet web access speed was at 2300/BPS in average. If it is in Internet rush hours, the performance could be slower.



The experimentation work verified the hypothesis of the overall approach. And it showed the efficiency of the distributed component hub's three tier architecture with LDAP implementation of the reuse-index and the reusable components naming server, particularly in 1,000 indexed components size range.

As for whether reusable software components should be stored in a central controlled storage or distributed in different local repositories, our experimentation shows advantages of distributed approach. In reality of enterprise Intranet, software components are built and stored locally for specific application domains [Beh98]. With our distributed component hub model, distributed storage can be easy to manage and control. And its operations and access are efficient.

5. Discussion and Conclusion

There have been considerable research work done in the area of software component retrieval and reuse management [Bro97, MOT97, MMM98, Beh98, Griss99]. Automated retrieval methods and components classification schema have been among the recent work's emphases so far.

[Beh98] presents an Internet-based software component information system(SCIS). It uses Oracle database schema for its reusable software components classification, which combines faceted classification [Pri91] and feature-oriented classification[Bor95] methods. Its search operations rely on Oracle's built-in search mechanism.

Profile matching and signature matching are described and discussed in [LG99], as retrieval methods. Software component is represented in component profile and signature based on component operation and property. The up-front cost for encoding the component profile and signature remains as a concern, particularly for a realistic reuse environment.

Component-based software paradigm implies progression and integration of reusable software components construction, repository and retrieval management, version control, and applications build and reuse. Software component reuse predisposes to domain and context dependent. Software use-case and design scenarios play an important role in component reuse process. The distributed component hub scheme with reuse-index and reusable component naming server systematically integrates the context-based categorization and hierarchical naming context into distributed repository management for more efficient retrieval. It's an effective augment to various retrieval operations, in addition to automated search mechanisms alone.

References

[AN95] ANSI/NISO Z39.50-1995, "ANSI Z39.50: Information Retrieval Service and Protocol", 1995. ftp://ftp.loc.gov/pub/z3950/

[Beh98] Anita Behle, An Internet-based Information System for Cooperative Software Reuse, Proceedings of the Fifth International Conference on Software Reuse, pp 236-245, June 2-5, 1998, Victoria, British Columbia, Canada.

[Boo87] Grady Booch, Software Components with Ada: Structures. Tools, and Subsystems, Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA, 1987.

[Bor95] J. Borstler, Feature-Oriented Classification for Software Reuse, Proceedings of the 7 th International Conference on Software Engineering and Knowledge Engineering (SEKE'95), pp.204-211, June, 1995, Rockville, ML, Knowledge Systems Institute, Skokie, IL, USA.

[Bra+98] Tim Bray et al., Extensible Markup Language(XML)1.0, W3C Recommendation, February 10, 1998. Http: //www.w3.org/TR/1998/REC-xml-19980210.

[BW99] Alan W. Brown and Kurt C. Wallnau, International Workshop on Component-Based Software Engineering, In Proceedings of the International Conference of Software Engineering'99, 1999, Los Angeles, CA, USA.

[Bro97] Shirley V. Browne and James W. Moore, Reuse Library Interoperability and the World Wide Web, Proceedings of the 1997 International Conference on Software Engineering, pp.684-691, May 17-23, 1997, Boston, Massachusetts, USA.

[CPT+97] James X. Ci, M. Poonawala, Wei-Tek Tsai, et al, ScmEngine: A Distributed Software

Configuration Management Environment on X.500, Proceedings of ICSE'97 SCM-7 Workshop, pp108-127, May 18-19, 1997, Boston, MA, USA.

[DK87] E.M. Dusink and J. Van Katwijk, Reflections on Reusable Software and Software Component, Proceedings of the Ada-Europe International Conference, pp.113-126, May 26-28, 1987, Stockholm.

[Griss99] Martin L. Griss, Architecting for Large-Scale Systematic Component Reuse, In Proceedings of the International Conference of Software Engineering'99, 1999, Los Angeles, CA, USA.

[HS97] Timothy A. Howes and Mark C. Smith, LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol, Macmillan Technical Publishing, 1997.

[Kai96] J. Bradford Kain, Components: The Basics: Enabling An Application or System to Be The Sum of Its Part, pp.64-69, Object Magazine, 6(2), April, 1996.

[LG99] Luqi and Jiang Guo, Toward Automated Retrieval for a Software Component Repository, Proceedings of IEEE Conference and Workshop on Engineering of Computer-Based Systems, March 1-12, 1999, Nushville, Tennessee, USA.

[MMM98] A. Mili, R. Mili and R. T. Mittermeir, A survey of software reuse libraries, pp349-414, Annals of Software Engineering, 5, 1998.

[MOT97] Nenad Medvidovic, Peyman Oreizy, and Richard N. Taylor, Reuse of Off-the-Shelf Components in C2-Style Architectures, Proceedings of the 1997 International Conference on Software Engineering, pp. 692-700, May 17-23, 1997, Boston, Massachusetts, USA.

[Pri91] R. Prieto-Diaz, Implementing Faceted Classification for Software Reuse, pp. 88-97, Communications of the ACM 34(5), ACM, New York, May, 1991.

[SHW98] Robert C. Seacord, Scott A. Hissam and Kurt C. Wallnau, Agora: A Search Engine for Software Components, Technical Report, CMU/SEI-98-TR-011, ESC-TR-98-011, August 1998.

[Szy98] Clemens Szyperski, Component Software Beyond Object-Oriented Programming, Addison-Wesley 1998.

[Weg89] Peter Wegner, Capital-intensive software technology, pp.43-97, Software Reusability, Vol. I: Concepts and Models, ACM Press, 1989.