Hyperrealised Semantic Search Design

Introduction

The design of our semantic search software is divided into the subsystems outlined in this document.

Hyperrealised Semantic Search Design

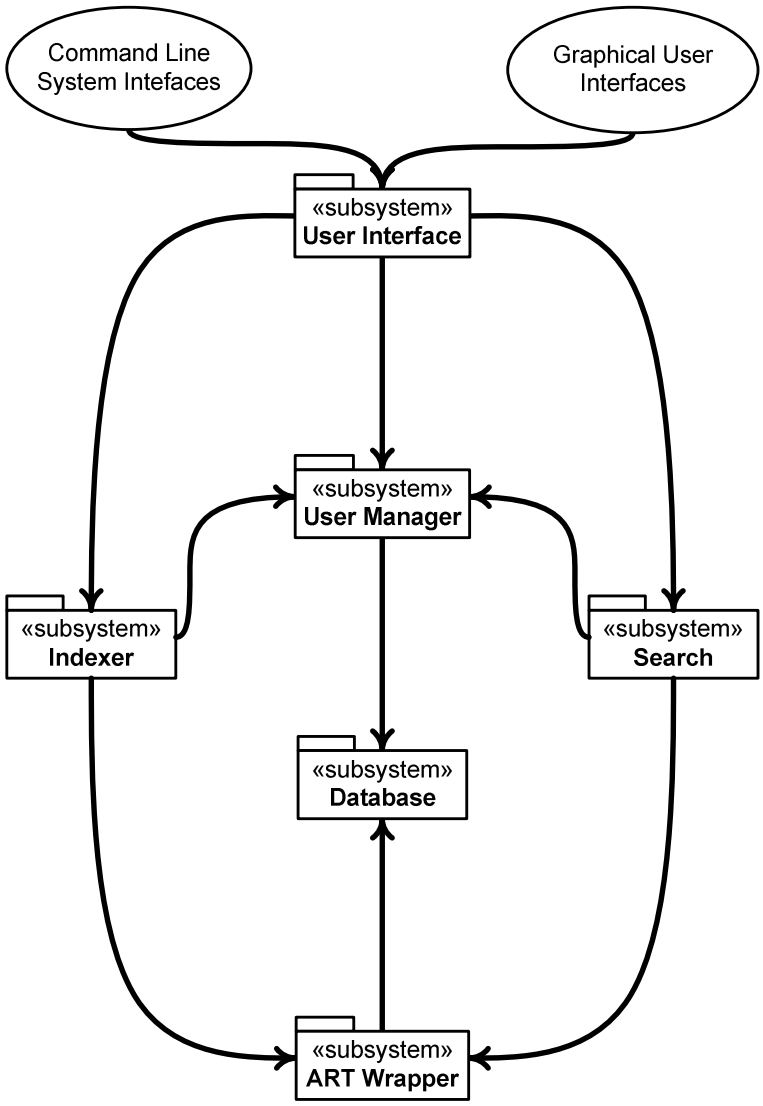
Contents

Hyperrealised Semantic Search Design

1. Overview

The design is broken down into 6 subsystems – The User Interface (front-end of the software), User Manager, Indexer, Search, ART Wrapper, and Database.

1.1. UML



Hyperrealised Semantic Search

Design

2. User Interface

2.1. Purpose

2.1.1. Provide a way to access and manipulate the facilities provided by the various other subsystems (the front-end of the system, the part the user interacts with).

2.1.1.1. Provide a way to add nodes and links from the WordNet database (import the WordNet database).

2.1.1.2. Provide a way to add nodes and links (or update nodes and links) from the compressed Wikipedia XML dump.

2.1.1.2.1. Decompress (BZIP2) and parse the XML file to discover articles.

2.1.1.2.2. Parse the Wikitext contained within the XML document to discover links between articles.

2.1.1.3. Provide a central point for user interfaces to interact with the system.

2.1.1.4. Log events as they occur.

2.2. Priorities

2.2.1. Create a parser for the Wikipedia BZIP2 XML dump to read the text and title of articles.

2.2.1.1. Find as many links as possible between articles.

2.2.1.2. Remove types of links that are detrimental to search results because they are not relevant enough.

2.2.2. Create a parser for the WordNet files.

2.2.3. Create simple XHTML user interface with PHP to allow user interaction through a web browser.

2.3. Outline

2.3.1. A third party software library will be used to provide access to java classes from PHP (http://php-java-bridge.sourceforge.net/).

2.3.2. A third party software library will be used to provide a means to decompress the BZIP2 compressed Wikipedia dump (http://commons.apache.org/sandbox/compress/).

2.3.3. The Wikipedia database will be downloaded as a BZIP2 compressed XML file (http://download.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2).

2.3.4. The Java SAX (Simple API for XML) library will be used to parse the Wikipedia XML file. The SAX library is an event based XML parser that allows efficient stream based parsing of XML files when the strategy pattern is used and allows parsing of very large XML files with a small memory footprint.

2.4. Issues

2.4.1. Some articles include an excessive number of links some of which are not all that relevant to the topic. To deal with this the nature of the link will be recorded (what type of link it is) to later be used in a filter to eliminate detrimental links.

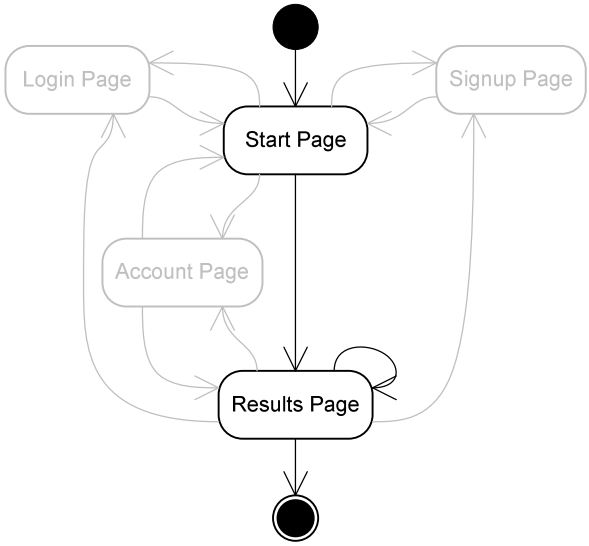
2.5. Graphical User Interface

2.5.1. Design

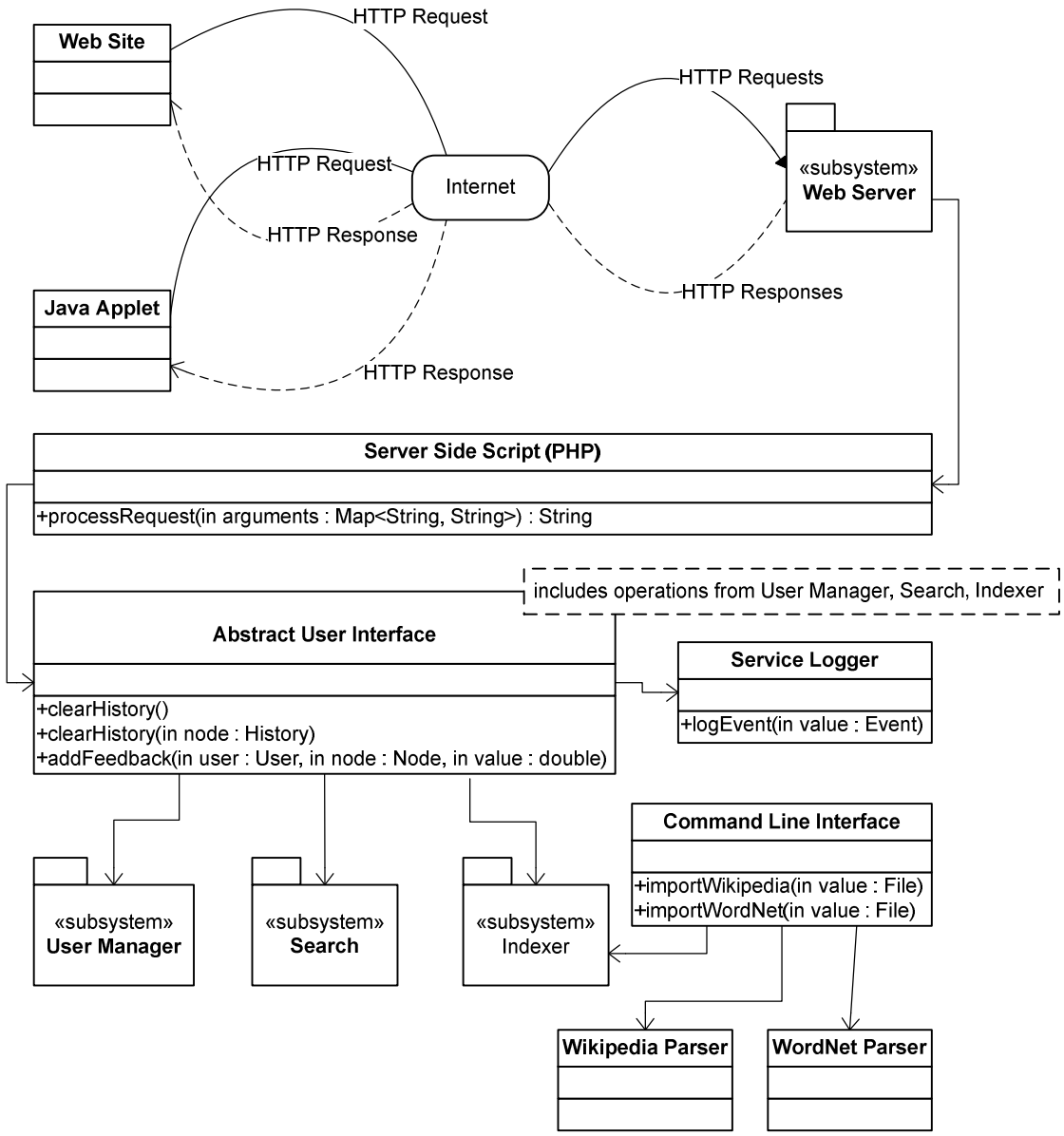
2.5.2. State Diagram

Hyperrealised Semantic Search

Design



2.6. UML


Web Site
HTTP Request
HTTP Requests
«subsystem»
Web Server
HTTP Request
Internet
HTTP Response
HTTP Responses
Java Applet
HTTP Response
Server Side Script (PHP)
+processRequest(in arguments : Map<String, String>) : String
includes operations from User Manager, Search, Indexer
Abstract User Interface
+clearHistory()
+clearHistory(in node : History)
+addFeedback(in user : User, in node : Node, in value : double)
Service Logger
+logEvent(in value : Event)
Command Line Interface
+importWikipedia(in value : File)
+importWordNet(in value : File)
«subsystem»
User Manager
«subsystem»
Search
«subsystem»
Indexer
Wikipedia Parser
WordNet Parser

3. User Manager

3.1. Purpose

3.1.1. Manage all operations regarding the creation and manipulation of user accounts.

3.1.2. Keep track of feedback given by users to assist in future searches.

3.2. Priorities

3.2.1. Keep track of user feedback.

3.2.2. Provide a means for users to create accounts and login to those accounts.

3.2.3. Provide a means for users to reset the passwords on their accounts or to delete those accounts.

3.3. Outline

3.3.1. Feedback is stored along with other nodes but can be marked with a weighting that will influence link weighting specific to a user.

3.3.2. An email system will be employed to provide facilities for sending emails to users. This will be a third party software library provided by SUN Microsystems (http://java.sun.com/products/javamail/index.jsp). The library will be used to send emails with a Gmail account (hyperrealised@gmail.com).

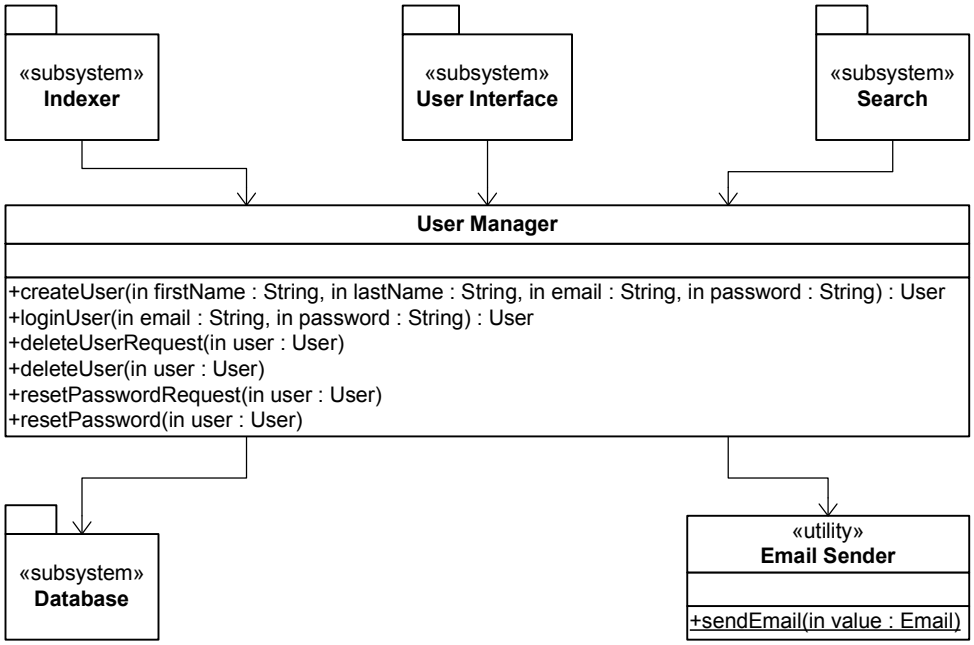
3.4. Issues

3.4.1. Link weighting for each user has to be stored somewhere temporally during calculation. The only way to do this within a reasonable amount of time is in memory. This will however create a further limitation on the depth of the search as execution time is no longer the only factor.

3.4.2. In the event that a user forgets their password they should be able to reset it. To achieve this functionality they can request that an email be sent to the email address registered to their account with a link that will allow their password to be reset to a new password which they will provide.

3.4.3. Users should have the option to delete their account. To prevent accidental deletion of accounts an email will be sent to the email address registered to the account providing a link to confirm that the account should be deleted.

3.5. UML


«subsystem»
Indexer
«subsystem»
User Interface
«subsystem»
Search
User Manager
+createUser(in firstName : String, in lastName : String, in email : String, in password : String) : User
+loginUser(in email : String, in password : String) : User
+deleteUserRequest(in user : User)
+deleteUser(in user : User)
+resetPasswordRequest(in user : User)
+resetPassword(in user : User)
«subsystem»
Database
«utility»
Email Sender
+sendEmail(in value : Email)

4. Indexer

4.1. Purpose

4.1.1. Parse documents for the purpose of extracting words to index them by so that they will be searchable by users.

4.1.2. Fetch documents that are located on the internet.

4.1.2.1. Add URLs of immediately linked documents to a low priority queue of pages to be fetched and parsed. This will perhaps provide more relevant documents to the user that requested the initial document be indexed and is a great use of the free high speed internet connection our server has. (very low priority)

4.1.2.2. Refresh local copy previously acquired documents in order to maintain an up-to-date index. (very low priority)

4.1.2.3. Determine what linked pages should be fetched in order to provide documents that are most relevant to the user. (very low priority)

4.1.3. Index uploaded files in the same way that downloaded files are.

4.1.4. Encode the content found within documents in order to save space. A compression algorithm will be used for this as it would reduce the space required to store the document without having to discard the contents and only keep the index. (extremely low priority as we are not short on hard drive space)

4.2. Priorities

4.2.1. Create a web crawler to fetch documents located on the internet. (low priority)

4.2.2. Create a generic parser that extracts words from documents. (low priority)

4.2.3. Determine how frequently the local copy of documents should be refreshed to prevent the local copy from becoming stale and without refreshing more frequently than required. (very low priority)

4.3. Outline

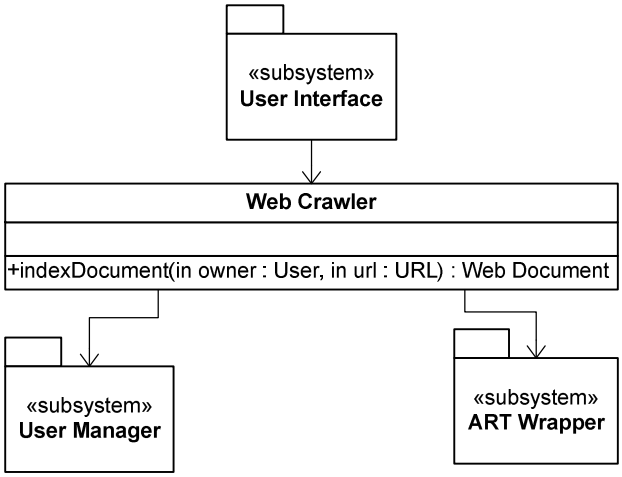
4.3.1. The queue of low priority documents to be fetched will be limited so that the queue only moves after hours to ensure we are considerate to other users of the shared internet connection.

4.3.2. The Java URL and URLConnection classes will be used to create a web crawler.

4.4. Issues

4.4.1. Crawling the web for more than a single document will affect the speed of the shared internet connection causing other users to have a slower connection. To work around this, only single documents will be fetched and indexed and other documents that are linked to these will be fetched and parsed after hours when the shared internet connection is in less demand.

4.5. UML



5. Search

5.1. Purpose

5.1.1. Provide a means to perform semantic search with no more than text for search terms.

5.2. Priorities

5.2.1. Use the facilities of the ART Wrapper to search for nodes based on the text they contain and allow those nodes to be used in the semantic search of all other nodes.

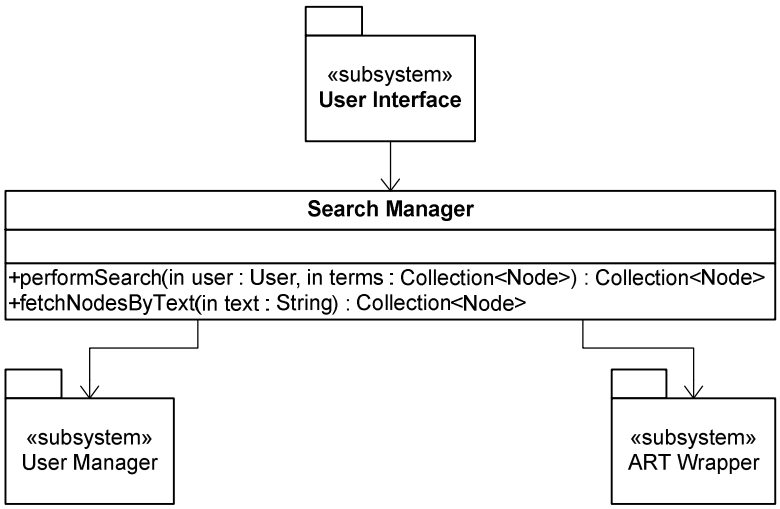
5.3. Outline

5.3.1. Not much more that a decoration of the User Manager and ART Wrapper.

5.4. Issues

5.4.1. Different types of nodes contain different types of content and yet all these nodes should be searchable with SQL through a single interface. To provide this functionality, nodes will contain a text field that will be used during searches as all nodes have at least some portion that can be represented as text. This text field will consist of no more than a phrase. For Documents this will be the title of the document and for Words it will be the word itself. Other types of nodes may not directly be searchable but instead will be accessed through the links they have to searchable nodes.

5.5. UML



6. Database

6.1. Purpose

6.1.1. Provide a means to persistently store, access and manipulate all nodes and links between those nodes and the content contained within them including:

6.1.1.1. Users (first name, last name, email address, password hash).

6.1.1.2. Documents (url if applicable, document content).

6.1.1.3. Words (the word itself, flags).

6.1.1.4. Queries (user, nodes).

6.1.1.5. Feedback (user, weight, document).

6.1.1.6. History.

6.1.2. Keep track of events that have occurred for future reference.

6.2. Priorities

6.2.1. Create JDBC connection to HSQLDB database engine.

6.2.2. Author the SQL required to create the tables.

6.2.3. Ensure the tables are indexed properly to allow optimal query time.

6.2.4. Author the SQL required to access and manipulate the tables.

6.2.5. Create high-level methods to access and manipulate tables.

6.3. Outline

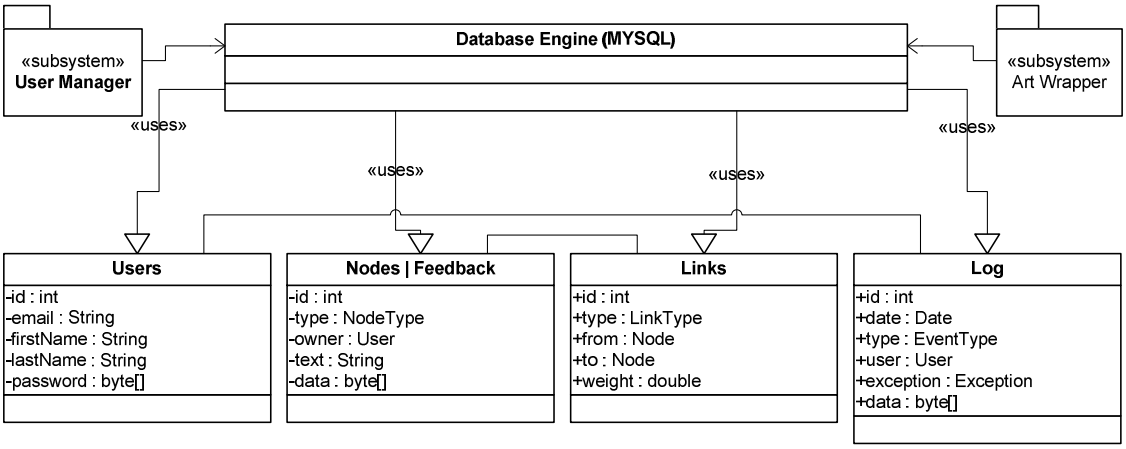
6.3.1. HSQLDB will be used as the database engine to make the software standalone and allow unit testing of code relating to the database.

6.3.2. Indexes will be created on the fields that are used as search terms.

6.4. Issues

6.4.1. The performance of the Nodes table may be affected by its abstract nature in that feedback would be stored along with all other nodes. To improve the performance of the table index and to simplify the queries made to the table it will be split in two.

6.5. UML


«subsystem»
User Manager
Database Engine (MYSQL)
«subsystem»
Art Wrapper
«uses»
«uses»
«uses»
«uses»
Users
-id : int
-email : String
-firstName : String
-lastName : String
-password : byte[]
Nodes | Feedback
-id : int
-type : NodeType
-owner : User
-text : String
-data : byte[]
Links
+id : int
+type : LinkType
+from : Node
+to : Node
+weight : double
Log
+id : int
+date : Date
+type : EventType
+user : User
+exception : Exception
+data : byte[]

7. ART Wrapper

7.1. Purpose

7.1.1. Provide a means for users to search a graph of nodes

7.1.1.1. Filter the nodes returned to the user to preserve the privacy of other users

7.1.2. Provide a means to manipulate a graph of nodes in the following ways:

7.1.2.1. Add nodes

7.1.2.2. Remove nodes

7.1.2.3. Add links

7.1.2.4. Remove links

7.1.3. Scale with the addition of tens of gigabytes of nodes from sources such as Wikipedia

7.1.3.1. Operate within a fixed amount of RAM and still be able to scale to larger numbers of nodes

7.1.3.2. Schedule the serialization and de-serialization of nodes and links to and from an SQL database

7.2. Priorities

7.2.1. Create a proxy graph of nodes and links that is backed by the database.

7.2.2. Create a generic splay tree.

7.2.3. Support scheduling of node serialization and de-serialization.

7.2.4. Create a means to search the graph structure using the ART library.

7.2.5. Create a means to manipulate the graph structure in the ways required by other subsystems.

7.2.6. Construct a proxy graph that filters nodes based on the user accessing the nodes.

7.3. Outline

7.3.1. The proxy design pattern will be used to create a data structure that is useable by the ART library as complete a graph of nodes

7.3.1.1. Allow virtual nodes that are de-serialized by the scheduler when they are requested by the ART library.

7.3.1.2. Hide nodes that should not be visible to the user that is manipulating or searching the graph.

7.3.2. A splay tree will be used to determine which nodes should be serialized to free up RAM

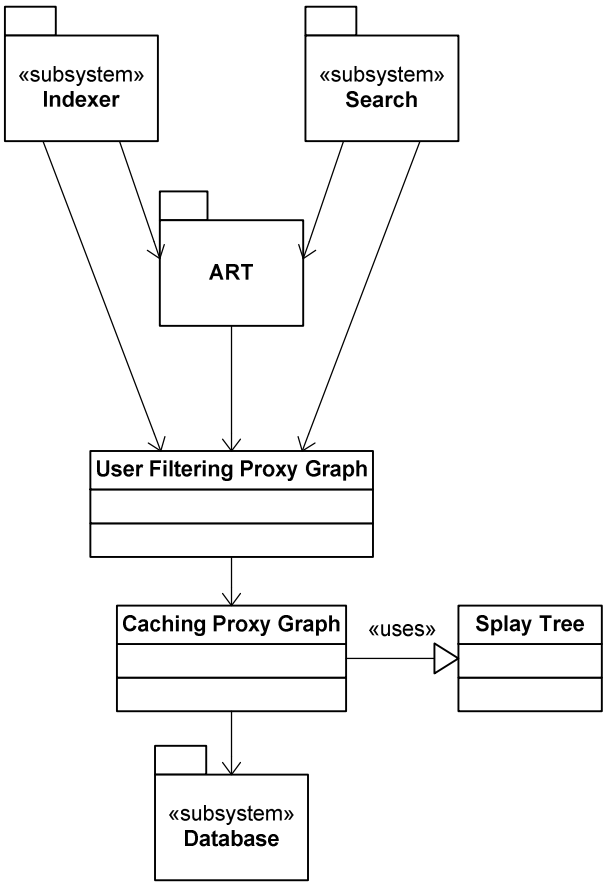
7.3.2.1. The splay tree will identify which nodes were accessed most recently.

7.3.3. JDBC (Java Database Connector) will be used to provide a connection to an SQL database.

7.4. Issues

7.4.1. Determining if more nodes can be de-serialized without breaching a RAM limit and causing an out-of-memory exception or if other nodes should first be serialized.

7.5. UML



8. Global Priorities

8.1. Minimal design

8.1.1. Create Wikipedia parser (2.2.1)

8.1.2. Create WordNet parser (2.2.2)

8.1.3. Create XHTML user interface (2.2.3)

8.1.4. Create JDBC connection to HSQLDB database (6.2.1)

8.1.5. Author SQL needed to create tables (6.2.2, 6.2.3)

8.1.6. Author SQL needed for data access and manipulation (6.2.4)

8.1.7. Create high-level methods to access and manipulate tables (6.2.5)

8.1.8. Create a proxy graph of nodes and links that is backed by the database (7.2.1)

8.1.9. Create a generic splay-tree (7.2.2)

8.1.10. Support scheduling of node serialization and de-serialization (7.2.3)

8.1.11. Create a means to search the graph structure using the ART library (7.2.4)

8.1.12. Create a means to manipulate the graph structure in the ways required by other subsystems (7.2.5)

8.2. Deferred components

8.2.1. Support user accounts (3.2.2)

8.2.2. Keep track of user feedback (3.2.1)

8.2.3. Support user account password reset (3.2.3)

8.2.4. Support user account deletion (3.2.3)

8.2.5. Create web crawler (4.2.1)

8.2.6. Create web document parser (4.2.2)

8.2.7. Support filtering of nodes that should not be accessible by the user (7.2.6)

8.2.8. Support refreshing local copies of web documents (4.2.3)

8.2.9. Support crawling further linked web documents (4.2.1)

8.2.10. Java RMI to allow distributed deployment.

9. Server Setup

9.1. Minimal

The server will consist of a single virtual machine running a web server (apache, php, php-java-bridge) and the semantic search software. The virtual machine has a 1.86 GHz Intel Core 2 Duo processor (only one core is allocated to the virtual machine) with 2 GB of RAM (less the virtual machine overhead) and an 80 GB hard drive.

9.2. Cluster (low priority)

The cluster will consist of a server and 93 slave computers. The server will consist of the web server and database engine. An instance of the semantic search software will run on the server and each of the slave computers. The instances on the slave computers will be connected to the instance on the server by means of Java RMI (remote method invocation) tunnelled through SSH allowing requests to the database engine and from the web server.

Requests from the web server will be allocated by the server to a slave computer for execution freeing the resources on the server to be used in processing multiple concurrent requests from the web server and to the database engine.

The slave computers are as follows:

Computer lab 1 (41 computers): Intel Core 2 Duo 1.86 GHz with 1 GB of RAM.

Computer labs 6 and 7 (52 computers): Intel Pentium 4 2.8 GHz with 512 MB of RAM.

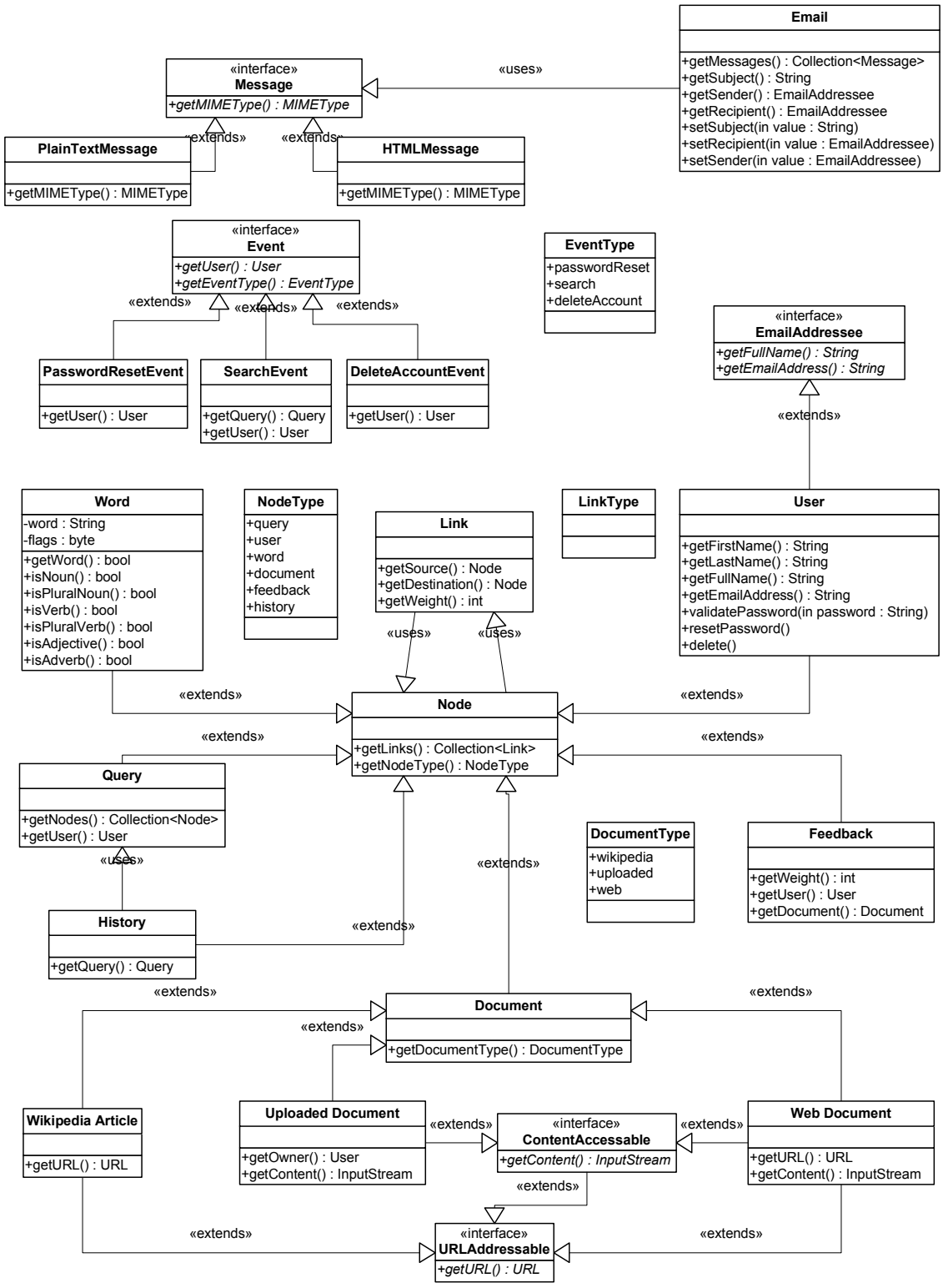
Combined system resources:

(1*1.86)+(41*2*1.86)+(52*2.8)= 299.98 GHz of CPUs.

(1*1.7)+(41*1)+(52*0.5)= 68.7 GB of RAM.

80 GB of HDD space.

10. Data Classes



11. Java Interfaces

```
1. public class SplayTreeMap<A, B> implements SortedMap<A, B> {
2. }
3. public interface Graph extends Iterable<Node> {
4.     public Iterable<Node> getNodesByText(String text);
5.     public void addNode(Node value);
6.     public void removeNode(Node value);
7.     public void addLink(Link value);
8.     public void removeLink(Link value);
9. }
10. public abstract class ProxyGraph implements Graph {
11.     private Graph mGraph;
12.     public ProxyGraph(Graph graph) {
13.         this.mGraph = graph;
14.     }
15. }
16. public class UserFilterProxyGraph extends ProxyGraph {
17.     public UserFilterProxyGraph(Graph graph) {
18.     }
19.     public void setUser(User user);
20. }
21. public class CachingProxyGraph implements Graph {
22.     private Map<String, Node> mNodeCache;
23.     public CachingProxyGraph(Database database) {
24.         mNodeCache = new SplayTreeMap<String, Node>();
25.     }
26. }
27. public class ARTProxyGraph extends art.Graph<Node, ARTProxyWeight, Node> {

28.     public ARTProxyGraph(final Graph graph) {
29.     }
30.     public void add(final Node from, final ARTProxyWeight weight, final Nod
    e to);
31.     public Map<Node, ARTProxyWeight> get(final Node from);
32.     public int size();
33.     public Iterator<Node> iterator();
34. }
35. public class ARTProxyWeight extends art.LinkWeight {
36.     public ARTProxyWeight(final double weight) {
37.     }
38.     public double weight();
39. }
40. public class ARTProxyMap implements Map<Node, ARTProxyWeight> {
41.     public ARTProxyMap(Node from) {
42.     }
43. }
44. public abstract class Node {
45.     public NodeType getNodeType();
46.     public User getOwner();
47.     public String getText();
48.     public Iterable<Link> getLinks();
49. }
```

```
50. public abstract class Link {
51.     public LinkType getLinkType();
52.     public Node getFrom();
53.     public Node getTo();
54.     public double getWeight();
55. }
56. public enum NodeType {
57.     Query, User, Word, Document, Feedback, History
58. }
59. public enum LinkType {
60. }
61. public interface EmailAddressee {
62.     public String getFullName();
63.     public String getEmailAddress();
64. }
65. public class Email {
66.     public Collection<Message> getMessages();
67.     public String getSubject();
68.     public EmailAddressee getSender();
69.     public EmailAddressee getRecipient();
70.     public void setSubject(String value);
71.     public void setSender(EmailAddressee value);
72.     public void setRecipient(EmailAddressee value);
73. }
74. public interface Message {
75.     public MIMEType getMIMEType();
76.     public String getContent();
77. }
78. public class HTMLMessage implements Message {
79. }
80. public class PlainTextMessage implements Message {
81. }
82. public abstract class Event {
83.     public EventType getEventType();
84.     public Date getDate();
85.     public User getUser();
86. }
87. public enum EventType {
88.     ResetPassword, Search, DeleteAccount
89. }
90. public class PasswordResetEvent implements Event {
91. }
92. public class SearchEvent implements Event {
93.     public Query getQuery();
94. }
95. public class DeleteAccountEvent implements Event {
96. }
97. public class Word extends Node {
98.     public boolean isNoun();
99.     public boolean isPluralNoun();
100.         public boolean isVerb();
101.         public boolean isPluralVerb();
102.         public boolean isAdjective();
```

```
103.          public boolean isAdverb();
104.      }
105.      public class User extends Node {
106.          public String getFirstName();
107.          public String getLastName();
108.          public void validatePassword(String password);
109.          public void resetPassword(String password);
110.      }
111.      public class Query extends Node {
112.          public Iterable<Node> getNodes();
113.          public User getUser();
114.      }
115.      public class History extends Node {
116.          public Query getQuery();
117.      }
118.      public class Feedback extends Node {
119.          public double getWeight();
120.          public User getUser();
121.          public Document getDocument();
122.      }
123.      public enum DocumentType {
124.          Wikipedia, Uploaded, Web
125.      }
126.      public abstract class Document extends Node {
127.          public DocumentType getDocumentType();
128.      }
129.      public class WikipediaArticle implements Document, URLAddressable {

130.          public String getTitle();
131.      }
132.      public class UploadedDocument implements Document, ContentAccessable
    {
133.      }
134.      public interface ContentAccessable {
135.          public InputStream getContent();
136.      }
137.      public class WebDocument implements Document, URLAddressable {
138.      }
139.      public interface URLAddressable implements ContentAccessable {
140.          public URL getURL();
141.      }
```