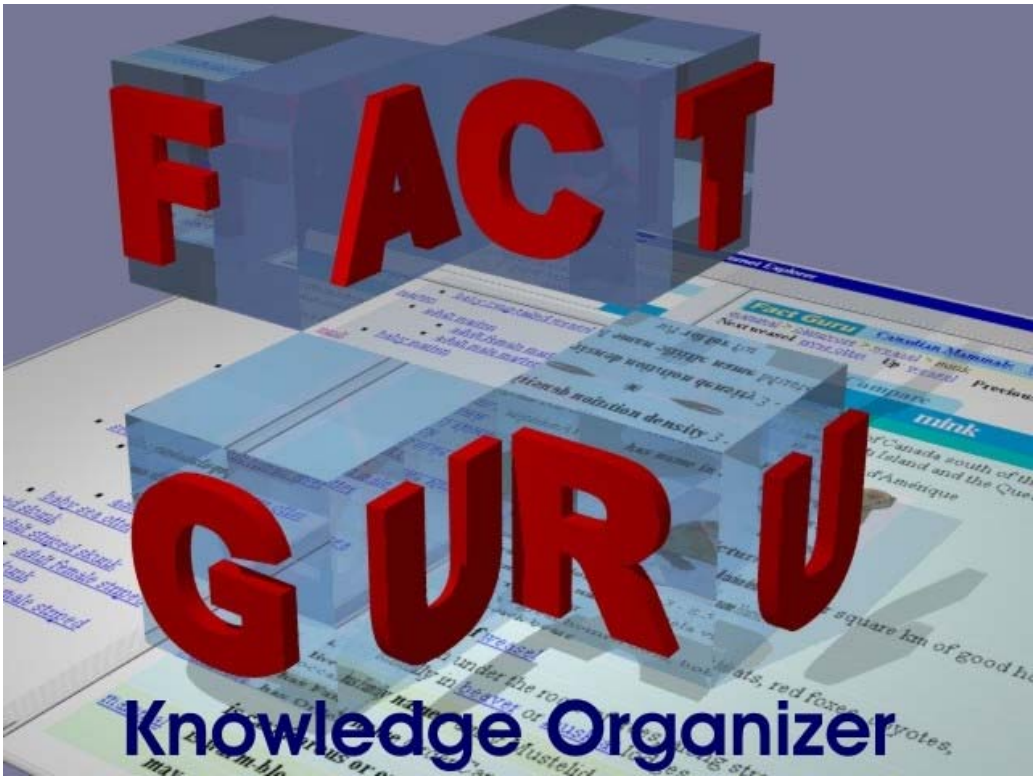


Fact Guru Requirements Analysis

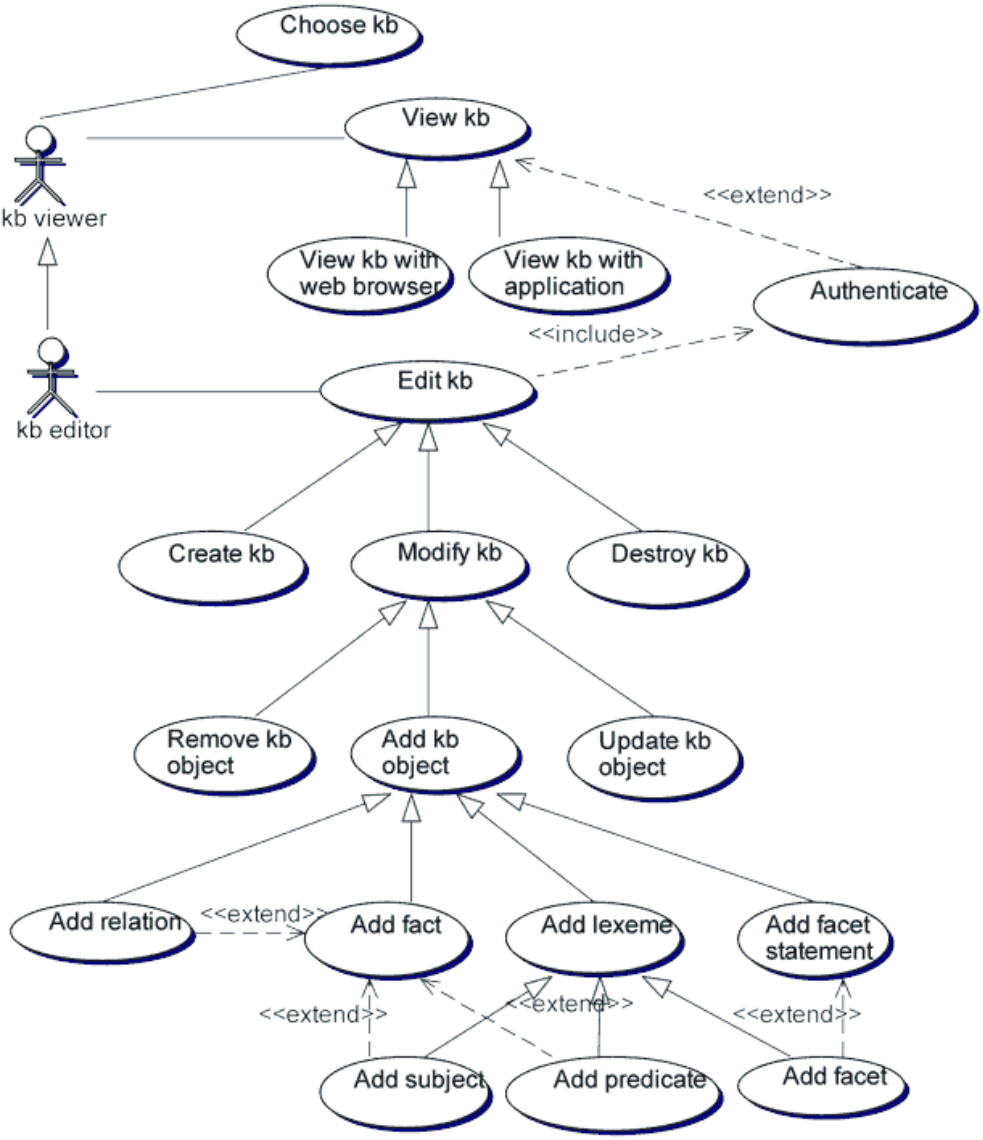
Author : John Talbot
Date : April 8, 2002
Purpose : Study the requirements for an efficient, flexible, portable and reliable knowledge base persistence mechanism. Then for each requirement make a tentative design decision. According to modern OO methodologies a software system is focused on implementing use-cases as a primary requirement specification. Therefore, attached to this email is a simplified UML use case diagram for the Fact Guru kb system.

There are two kinds of Fact Guru user :

1. **Kb viewer** is a kind of person
   has role browses kb or finds a kb using a search engine
   has commitment casual user reluctant to download anything
   has non-functional requirement runs within most browsers without downloading anything
2. **Kb editor** is a kind of **kb viewer**
   has role entering knowledge into kb
   has commitment serious user willing to download the our kb editor/viewer Java classes and downloading the knowledge base
   has non-functional requirement runs within the latest browser

- *The functional requirement :* A knowledge base must be remotely editable by kb editors. Note, the plural, a kb must allow concurrent editing and be scalable.
- *The issues :* Our Java KO tool is out of date relative to the user friendly web export format. The tool is difficult to work with, has poor usability, has bugs and requires Java 1.3
- *The design decision :* Create a server based version of Fact Guru and make it behave exactly like the user friendly web export but run within any web browser. The architecture will be based on J2EE and make use of servlets, EJB and web services (XML, XSLT and SOAP). For the mobile capabilities we will make use of J2ME using MIDP+XML to support portables devices having a wireless connection to the net such as Java compatible cell phones, pagers and Palms.

Fact Guru Use Case Diagram



(created using Together Community 5.5)

Requirements (in yellow)

Persistence : Store and publish a valid kb in a portable data format. Definitions: 1. Portability: Easy to copy and install on any platform, does not require database 2. Validity : Prevent circularity, no children of instances, etc... 3. Storage : Save the facts in a format which can be edited and version controlled 4. Publish: allow read access over the internet or HTTP
1. **Portability** is achieved by using XML as the data format. The consequences of choosing XML is that the kb must be small enough to fit entirely into RAM, this is not a problem with current computers. For example (scenario): the astronomy kb has 18000 facts and can be entirely loaded into a machine with 128 MB RAM) **Portability** can be further increased by restricting the XML data to a single file by including the DTD in the header of the file. We decided to drop database support, although databases can support extremely large kbs and concurrent authoring, the database servers are expensive and not widely available, also because kbs encoded as database tables are notoriously difficult to share. Note: The current system supports databases, this support will remain, however it will not be the default persistence mode, nor will it be extended or optimized. 2. **Validity** is achieved by satisfying integrity constraints before the data is stored as XML. This includes assigning unique ids to all major kb objects. Referential integrity is enforced for all kb objects which are represented as rows in a database table have an id :

- kb objects with id : SUBJECT, PREDICATE, FACET, STATEMENT, FACETSTATEMENT, RELATION
- kb objects without id : PREFIX, VERB, POSTFIX, COMPLEMENT, REST
- non-kb objects don't have an id : KNOWLEDGEBASE, PREDICATES, FACETS, RELATIONS, SUBJECTS, NAME, VERBPHRASE

3. **Storage** is achieved by recording associated information in the hidden attributes of kb objects, attributes such as sense, category, ordering, dimension etc... Version control is achieved by including the following attributes on all kb objects with an id : createdBy, modifiedBy, modifiedWhen, userComment
4. **Publishing** is achieved in several ways (all of which can coexist)
   - (a) provide an applet to parse and view the kb encoded as XML data
   - (b) generate web pages which contain the facts for a given subject, the HTML code generation process can be static or dynamic depending on client requirements :
     - (b1) **Static generation** : The appropriate HTML code for the entire kb is created only once and stored on a high capacity web server.
       - Disadvantages:
         1. The storage requirement for these pages can easily exceed the size of the XML file by a factor of 50. This is a problem with most ISP which limit the size of web sites to about 10 MBytes. (for example (scenario) : the astronomy kb occupies 60 MBytes)
         2. The creation phase is computationally expensive for the server, therefore the kb update frequency should correspondingly be low.
       - Advantage:
         1. The server can support many orders of magnitude more clients than dynamic generation process
         2. The kb can be published to servers that do not support running Java
         3. The server does not need to compute anything, only transfer web pages.
     - (b2) **Dynamic generation :** The HTML code is created by the server only when a client demands the facts about a subject.
       - Disadvantage:
         1. The server cannot support as many clients as the static generation process because it is computationally expensive to dynamically create fact pages.
         2. Most commercial servers cannot support the running of Java programs that are required to dynamically create the HTML.
       - Advantage:
         1. Very little server storage is required
         2. The fact page is always up to date, this supports a high knowledge base update frequency.

Efficiency: Rapidly load a knowledge base over the internet into an application.

- Efficiency is achieved by designing the XML DTD to map as closely as possible to the native kb object representation
- Efficiency is increased by constraining the XML data to a single file and including the DTD in the header requiring only one network connection per knowledge base.
- Efficiency is further increased by compressing the file using a ZIP tool. (XML compressed very well because there is a lot of repetition in the TAGS)
- Efficiency is further increased by including the XML file within the compressed JAR file containing the knowledge organizer applet. The special case of partial transmission failure.

Reliability : Load a partial knowledge base and despite the transmission failure; display the most important subjects and facts (but forbid editing to prevent corruption). This is especially important for very large knowledge bases because the probability of failure increases with file size.

- To minimize the effects of incomplete transfer of the XML file, the subjects must be ordered by depth so that the most important subjects in the hierarchy are transferred first.
- Another benefit of this ordering is that an initial outline of the concept hierarchy could be displayed in a Java application while the remaining subjects and facts are loaded concurrently.

Support browsing : View the knowledge base in a web browser without loading a Java application and using only the data file which encodes the kb. Browsing is not the same as publishing, because much of the information is elided for simplicity. Browsing supports a simple linear list of : subject+verbphrase+complement+rest (+facetstatements).

- Browsing is achieved by using XSL to view the XML file in a compatible browser which supports XSL style sheets. (Java is not required).
- The structure of the XML file is constrained to support simple XSL style sheets.
- Browsing the kb in bar delimited file format is not acceptable because it has not been validated. Also, the text format is much slower to load than a the XML file because it must be parsed and imported, therefore it should not be made public.

Flexibility : Ensure that the structure and content of the storage format is flexible so that the definitions can be refined (e.g. to rationalize it with developing standards).

- Make use of XML Schema, a developing standard which allows the structure and content of an XML file to be defined in much more powerful way than the deprecated DTD definition language.
- We can uncouple our XMLKBServer Java code from the structure of the XML file by adopting a 'stategy' design pattern. The structure of the XML file is defined in the XML Schema file and is determined at run time, not hard coded. The XMLKBServer implementations for inserting, updating and removing KBObjects obtain the position of the elements which map to these objects from a 'stategy' object which is created based on data in the XML Schema file and passed to the server. The strategy object hides the details of the XML file structure by providing a simplified 'facade' between the XML file and the KBServer. The role of the strategy object is to wrap the XML Schema file and provide structure independent access to KBObjects encoded as elements in the XML file. The consequences of this approach are :
    1. The structure and content of the XML file can evolve in parallel with developing standards and could be extended to satisfy the future requirements of our suite of knowledge management tools.
    2. Allow different applications to share the same set of XML files even if one or both applications changes the structure and content of the file as they evolve.
    3. The application is designed around the XML Schema, it is not hard-coded to the structure of the XML file. The Schema is written in XML and can be parsed using the same parser used to parse the content of the XML file.
    4. XML Schema supports datatypes and their restriction or extension, inheritance, abstraction and numerous other features which allow a much higher degree of precision when used to validate kbs or docubases encoded as XML files.
    5. XML Schema also supports uniqueness and primary and foreign keys typically used in databases, (and in our current database kb persistence format)
    6. XML Schema also supports more poweful mixed definitions which are useful for docubases because tags are 'mixed' with text.
    7. XML Schema provides a facility to allow Java class file to be associated with XML elements by using the annotation/appinfo/bind/class element. This would allow a further decoupling of the XML file from the XMLKBServer by allowing the XML file itself to decide what KBObjects map to each element.
    8. XML Schema is much better than DTDs because DTDs are not written in XML (it is EBNF) and DTD is a much less powerful way to define the structure and content of an XML file. The DTD will be included only for the sake of compatibility to support validity checking and XSL browsing using an XSL compatible browser such as IE5+. (Note: IE5+ does not support XML Schema files).
- Wratko Hlavina, the designer of the JDBCKBServer, attempted to use such a 'stategy' based design pattern by using the JDBC metadata for the database tables which would allow a decoupling of the JDBCKBServer from the structure and content of the tables. Unfortunately, the JDBC drivers provided at that time by the major database vendors (Oracle and Access) did not support meta data queries.
- Make use of XML data binding frameworks to automate much of the task of mapping XML to Java.
    - JAXB : Java API for XML Binding. Uses the DTD to map XML to Java objects. It is an official package developed by Sun.
    - CASTOR is more mature than JAXB and support sXML Schema which is much more powerful than DTD. Unfortunately it is not part of the Java community process.

Docubase support : A docubase is a combination of a knowledge base and a document. A docubase editor has the role of editing a collection of source sentences in loose English format and the option to associate each sentence with one or more knowledge base facts within the same data storage format. A fact can have a source sentence. A sentence can have zero or more facts associated with it. A sentence and a fact have equal importance. i.e. When a sentence is deleted any facts that were derived from it remain in the kb. When a fact is deleted the sentence from which it was derived remains in the document.

- Although a docubase editor is planned as an important component in our suite of knowledge management tools, we discuss it now to incorporate 'hooks' into our framework at an early stage of design to facilitate later implementation of such a tool.
- Support for docubases is realized by using XML Schema (see **flexibility** requirement).

Detailed design

Mapping

- The mapping from kb structure to XML file structure is based on the format of the bar-delimited text import file but with PREDICATE, FACET and RELATION as separate lists of XML tags.
- We originally planned to use hierarchies of XML SUBJECT TAGS to represent the concept hierarchy however this approach suffered from excessive duplication. (also it cannot recover from partial loading failure)
- When mapping kb objects to XML tags care must be take to 'escape' the following characters : < must be mapped to <, > to > and & to &. And the reverse mapping must be performed when the XML data is converted back into knowledge base objects.

Attributes:

- We constrain attributes to be items which do not have to be displayed when browsing the kb. For example : the predicate attribute in VERBPHRASE tag or the id attribute.
- To satisfy this requirement we design DTD to facilitate browsing of the kb using simple XSL style sheets which can be interpreted and displayed by XML compatible browsers such as Internet Explorer 5 or later version.
    - (a) An exception is made for FACETSTATEMENT facet attribute to avoid FACET duplication

- (b) An exception is also made for all the RELATION attributes : A flat list of relations is not useful; relations should be shown as a tree. Browsing the XML file for a list of relations is not a typical use case.
- (c) We are undecided about using predicates as the relation 'type' attribute or to simply put the relation as a string. For example :

```
<RELATION id="456" parent="677" child="123" type="is a kind of"/>
or
<RELATION id="456" parent="677" child="123" type="45"/>
```

where 45 maps to the 'is a kind of' predicate. The predicate reference method is more compact, reliable and powerful

The DTD

The DTD describes the structure/grammar of the XML tags which represent encoded kb objects. Plain English description of DTD :

1. A list of PREDICATE
2. A list of FACET
3. A list of RELATION
4. One or more SUBJECT within a KNOWLEDGEBASE
5. Zero or more STATEMENT within a SUBJECT
6. zero or more FACETSTATEMENT within a STATEMENT

Example of an XML encoded knowledge base:

```
<KNOWLEDGEBASE name="astronomy">
  <PREDICATES>
    <PREDICATE id="45" sense="0" inheritance="" inheritsOver="" inverse="45" category="system object">is a kind of</PREDICATE>
    <PREDICATE id="56" sense="0" inheritance="" inheritsOver="" inverse="56" category="system object">is a generalization of</PREDICATE>
    <PREDICATE id="46" sense="0" inheritance="" inheritsOver="" inverse="57" category="system object">is a part of</PREDICATE>
    <PREDICATE id="57" sense="0" inheritance="" inheritsOver="" inverse="46" category="system object">has part</PREDICATE>
    <PREDICATE id="47" sense="0" inheritance="" inheritsOver="" category="system object">is a subtopic of</PREDICATE>
  </PREDICATES>
  <FACETS>
    <FACET id="3" sense="0" inheritance="" inheritsOver="" inverse="" category="system object" modifiedWhen="15 June 2001, 23:14:21">has source</FACET>
    <FACET id="4" sense="0" inheritance="" inheritsOver="" inverse="" category="system object" modifiedWhen="15 June 2001, 23:14:21">has URL</FACET>
  </FACETS>
  <RELATIONS>
    <RELATION id="456" parent="677" child="123" type="is a kind of" dimension="" ordering="" createdBy="" modifiedBy="" modifiedWhen="" userComment=""/>
    <RELATION id="457" parent="12" child="44" type="is a part of"/>
  </RELATIONS>
  <SUBJECTS>
    <SUBJECT id="1" sense="0" category="system object" createdBy="" modifiedBy="" modifiedWhen="" userComment="">
      <NAME>kbTop</NAME>
    </SUBJECT>
    <SUBJECT id="123" sense="0" category="instance">
      <NAME>animal</NAME>
      <STATEMENT id="5467"> (optional)
        <VERBPHRASE predicate="45">    // predicate is an attribute because it does not have to be displayed
          <PREFIX></PREFIX> (optional)
          <VERB></VERB>
          <POSTFIX></POSTFIX> (optional)
        </VERBPHRASE>
        <COMPLEMENT>kbTop</COMPLEMENT>
        <REST></REST> (optional)
        <FACETSTATEMENT id="356" facet="34">John Talbot</FACETSTATEMENT> (optional)
        <FACETSTATEMENT id="864" facet="54">http://www.site.uottawa.ca</FACETSTATEMENT> (optional)
      </STATEMENT>
      <STATEMENT id="5467"> (optional)
        <VERBPHRASE predicate="45">
          <PREFIX></PREFIX> (optional)
          <VERB></VERB>
          <POSTFIX></POSTFIX> (optional)
        </VERBPHRASE>
        <COMPLEMENT>kbTop</COMPLEMENT>
        <REST></REST>
        <FACETSTATEMENT id="356" facet="34">John Talbot</FACETSTATEMENT>
        <FACETSTATEMENT id="864" facet="54">http://www.site.uottawa.ca</FACETSTATEMENT>
      </STATEMENT>
    </SUBJECT>
  </SUBJECTS>
</KNOWLEDGEBASE>
```

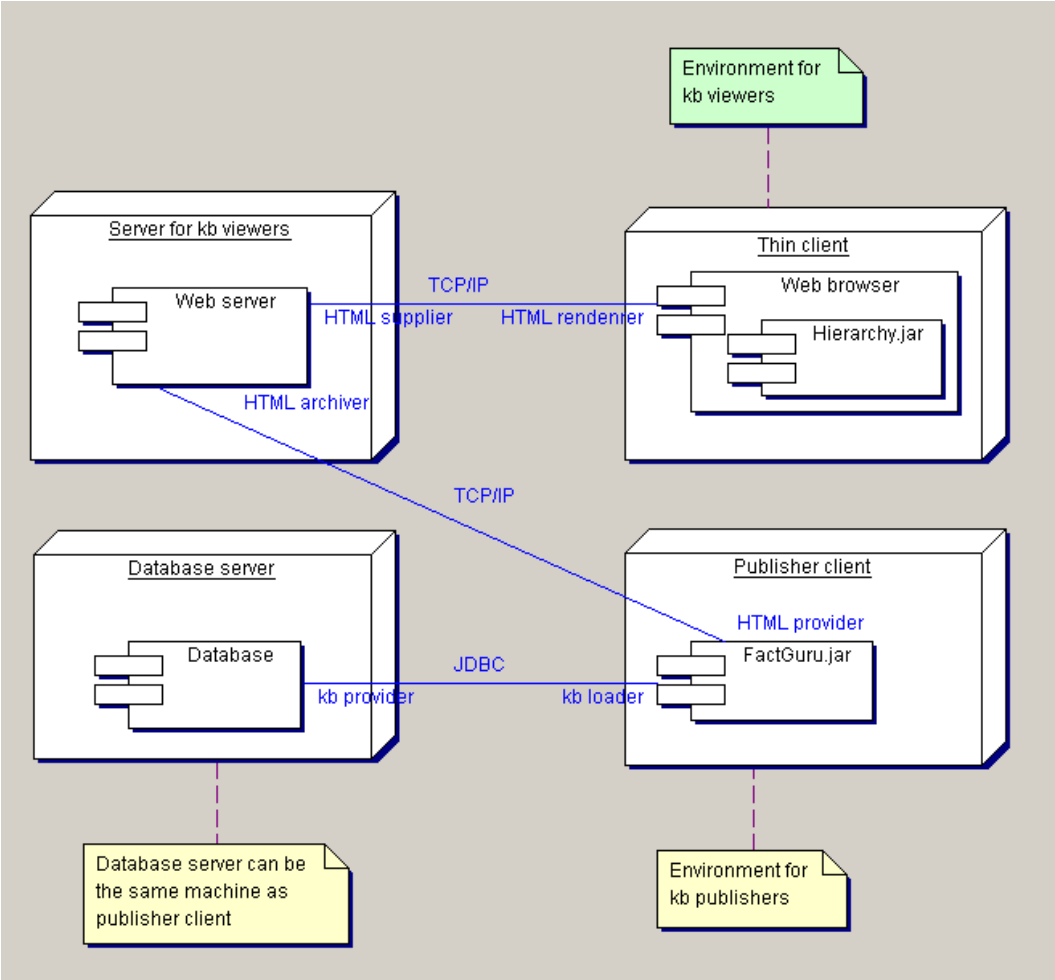
Schema based approach

```
  <xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
              targetNamespace = "http://factguru.com/schemas/kb"
              xmlns:kb = "http://factguru.com/schemas/kb">

    <xsd:element name="docubase">
    <xsd:complexType>
      <xsd:all>
        <xsd:element ref="document" minOccurs="0" maxOccurs="1">
        <xsd:element ref="knowledgebase" minOccurs="0" maxOccurs="1">
      </xsd:all>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="document">
```

```
    <xsd:all>
      <xsd:element ref="sentence" minOccurs="0" maxOccurs="unbounded">
    </xsd:all>
  </xsd:element>

  <xsd:element name="sentence">
    <xsd:complexType>
      <xsd:sequence>
    <xsd:complexType>
  </xsd:element>

  <xsd:element name="knowledgebase">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="predicates" minOccurs="0" maxOccurs="1">
        <xsd:element ref="facets" minOccurs="0" maxOccurs="1">
        <xsd:element ref="relation" minOccurs="0" maxOccurs="1">
        <xsd:element ref="subjects">
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="predicates">
    <xsd:complexType>
      <xsd:element ref="predicate" minOccurs="0" maxOccurs="unbounded">
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="facets">
    <xsd:complexType>
      <xsd:element ref="facet" minOccurs="0" maxOccurs="unbounded">
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="relations">
    <xsd:complexType>
      <xsd:element ref="relation" minOccurs="0" maxOccurs="unbounded">
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="subjects">
    <xsd:complexType>
      <xsd:element ref="subject" minOccurs="0" maxOccurs="unbounded">
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="kbObject">

  </xsd:complexType>

  <xsd:complexType name="kbIDObject">
    <xsd:complexContent>
      <xsd:extension base="kbObject">
        <xsd:attribute name="id" type="idType" use="required"/>
        <xsd:attributeGroup ref="audit" use="optional"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:attributeGroup name="audit">
    <xsd:attribute name="createdBy" use="optional"/>
    <xsd:attribute name="modifiedBy use="optional"/>
    <xsd:attribute name="modifiedWhen type="date" use="optional"/>
    <xsd:attribute name="userComment" use="optional"/>
  </xsd:attributeGroup>

  <xsd:unique name="idUnique">
     <xsd:selector xpath="kbIDObject"/>
     <xsd:field xpath="@id"/>
  </xsd:unique>

  <xsd:key name="idKey">
    <xsd:selector xpath="kbIDObject"/>
    <xsd:field xpath="@id"/>
  </xsd:key>

  <xsd:complexType name="lexeme">
    <xsd:complexContent>
      <xsd:extension base="kbIDObject">
        <xsd:attribute name="name">
        <xsd:attribute name="sense">
        <xsd:attribute name="category">
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="subject">
    <xsd:extension base="Lexeme">

      <xsd:attribute name="instance" type="boolean" default="false"/>
    </xsd:extension>
  </xsd:complexType>

  <xsd:complexType name="predicate">
    <xsd:complexContent>
      <xsd:extension base="kbIDObject">

        <xsd:simpleType /> ???

        <xsd:attribute name="inheritance" type="inheritance" default="full">
        <xsd:attribute name="inheritsOver" type="hierarchy" default="is a kind of">
```

```
        <xsd:attribute name="inverse">
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:simpleType name="inheritance">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="full">
      <xsd:enumeration value="predicate only">
      <xsd:enumeration value="none">
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="hierarchy">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="is a kind of">
      <xsd:enumeration value="is a part of">
      <xsd:enumeration value="is a topic of">
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="StatementPart">

  </xsd:complexType>

  <xsd:complexType name="prefix">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="StatementPart">
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:complexType>

  <xsd:complexType name="verb">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="StatementPart">
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:complexType>

  <xsd:complexType name="postfix">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="StatementPart">
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:complexType>

  <xsd:complexType name="complement">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="StatementPart">
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:complexType>

  <xsd:complexType name="rest">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="StatementPart">
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:complexType>

  <xsd:element name="relation">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="kbIDObject">
          <xsd:attribute name="parent" type="subjectID"/>
          <xsd:attribute name="child" type="subjectID"/>
          <xsd:attribute name="type"/>
          <xsd:attribute name="dimension"/>
          <xsd:attribute name="ordering"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xds:element>
  <xsd:keyref name="idKeyRef" refer="idKey">
    <xsd:selector xpath="relation"/>
    <xsd:field xpath="@parent"/>
    <xsd:field xpath="@child"/>
  </xsd:keyref>

</xsd:schema>
```

Current Thin Client deployement diagram (Web export of Knowledge Bases)



(created using Together Community 5.5)

Proposal for Text Analyzer Table Normalization

Author : John Talbot
Date: April 11, 2002

Purpose : The original Text Analyzer *sentences* table should be normalized into as many as 11 separate tables to improve search efficiency by allowing indexing of patterns and words. Another issue is the scattered nature of the patterns. The original pattern knowledge is currently stored in three separate incompatible locations :

1. Hardcoded in the Perl scripts used for pre-processing
2. Hardcoded in the Visual Basic 6.0 code
3. Stored as human readable natural language pattern descriptions in the Text Analyzer overview documents

A future version of the Text Analyzer will support dynamic/on-the-fly/real-time pre-processing of sentences from a document or the web. We facilitate this by unifying and explicitly encoding all this pattern knowledge into related tables. This will also simplify upgrading the patterns when researchers discover more patterns. We can also encode or parameterize the VB6 TA pattern refinements procedures into tables, however this is left for a future date.

Normalization Procedure

Currently several sentence table columns are overloaded with excessive content. For example, the tagged sentence column should be normalized into as many as five separate tables. Also the question type and pattern type columns can be factored into six separate tables. Each sentence has a collection of tagged words, each word has a stem, a syntactic tag and a part of speech tag. The relationship between sentences and tagged words is Many-to-Many Bi-directional; which can be represented using three tables : **sentence**, **stemmed_word** and **occurrence**. The types of syntactic tag and a part of speech tags can be factored into two separate tables : **syntactic_tag** and **part_of_speech_tag**. The occurrences table would hold a Many-to-One Unidirectional relationship to each of these tables. Each sentence also has a set of question types and pattern types found in the sentence; this can be represented as two relationships tables : **sentence-pattern** and **sentence-question** which hold a Many-to-Many Bi-directional

relationship with the **pattern** and **question** tables. The relationship between the question type and pattern type table is Many-to-Many Bi-directional and is stored in the **question-pattern** table. The pattern type table can be further normalized into One-to-Many Unidirectional relationship between a pattern type and the word strings that match it by creating yet another relationship table **pattern-patstring**.

Advantages

1. The current TA search strategy has linear time performance. Each question/answer session for a given word performs a highly inefficient linear search. By normalizing the tables, and by creating indexes we could greatly optimize performance.
2. The new table format can be transparently mapped to Enterprise JavaBeans. Entity beans using container managed persistence (CMP EJB) can be used to persist each table. There is a one-to-one correspondence between an entity bean and a table. Each table row represents an instance of an entity bean. The complex table joins and queries required to implement the relationships using raw SQL are handle automatically in the EJB framework as relationship fields.

References

- Kavanagh, J., Talbot, J., Skuce, D. : 1999, *The Text Analyzer*, LAKE Lab, University of Ottawa.
(id: kavanagh pass: chapters)

Detailed Table Structure

The Old Format

This is a row from the current *sentences* table :

sentences											
id	**doc**	**Sent**	**Par**	**Head**	**PrevHP**	**List**	**ListHP**	**Sentence**	**tagged sentence**	**questions**	**patterns**
345	5	13	6	0	4	I	12	Conductivity is measured by an instrument called a solubridge or conductivity meter.	Conductivity ~conductivity `S ^S|is ~be `X ^V|measured ~measure `V ^2|by ~by `A ^R|an ~an `D ^T|instrument ~instrument ` ^S|called ~call `V ^2|a ~a `D ^T|solubridge ~solubridge `O ^S|or ~or ` ^&|conductivity ~conductivity `O ^S|meter ~meter `O ^S|. ~ ` ^	-1-4-5-	-34-67-

The New Format

- The above *sentences* table can be normalized by decomposing the tagged sentence column into 4 separate tables
    1. **sentence** : The natural language sentence and structural information such as document, paragraph, header level etc...
    2. **stemmed_word** : unique list of stem words
    3. **occurrence** : implements the Many-to-Many Bidirectional relationship between a sentence and a stemmed word; it contains specific occurrences of the stemmed words in the sentence and their syntactic tag and a part of speech tags
    4. **syntactic_tag** : subject, verb, etc...
    5. **part_of_speech_tag** : noun, pronoun, adjective, etc...
- Each table has a primary key column called id, which is a unique number within that table.
- Each table has a unique colour code
- Tables which are not referred to are colour coded grey
- Relationship columns which represent foreign keys to other tables are colour coded according to table type

sentence								
id	**doc**	**Sent**	**Par**	**Head**	**PrevHP**	**List**	**ListHP**	**content**
345	5	13	6	0	4	I	12	Conductivity is measured by an instrument called a solubridge or conductivity meter.

occurrence						
id	**sentence**	**pos**	**word**	**stemmed_word**	**syntactic_tag**	**part_of_speech_tag**
567	345	0	Conductivity	0	3	8
568	345	13	is	1	1	17
569	345	16	measured	2	2	14
570	345	25	by	3	9	15
571	345	28	an	4	10	7

572	345	21	instrument	5	0	8
573	345	32	called	6	2	14
574	345	39	a	7	10	6
575	345	41	solubridge	8	5	8
576	345	52	or	9	0	4
577	345	55	conductivity	10	5	8
578	345	68	meter	11	5	8
579	345	73	.	12	0	0

stemmed_word	
id	**stem**
0	conductivity
1	be
2	measure
3	by
4	an
5	instrument
6	call
7	a
8	solubridge
9	or
10	conductivity
11	meter
12	.

syntactic_tag		
id	**code**	**tag**
0	`	none
1	`X	aux. verb
2	`V	main verb
3	`S	subject
4	`F	formal subject
5	`O	direct object
6	`I	indirect object
7	`P	Pcompl-S
8	`p	Pcompl-O
9	`A	adverbial
10	`D	determiner

part_of_speech_tag		
id	**code**	**tag**
0	^	none
1	^A	adjective
2	^B	abbreviation
3	^D	adverb
4	^&	coord. conjunction
5	^C	subord. conjunction
6	^T	determiner
7	^I	infmark (infinitive)
8	^S	singular noun
9	^L	plural noun
10	^G	genitive noun (possessive)
11	^N	noun, singular or plural
12	^#	number
13	^1	PCP1 -ing

14	^2	PCP2 -ed
15	^R	preposition
16	^P	pronoun
17	^V	verb

The questions and patterns columns of the original *sentences* tables can be decomposed into 5 tables.

1. **sentence-pattern** : Many-to-Many relationship between sentences and pattern types
2. **sentence-question** : Many-to-Many relationship between sentences and question types. This table is redundant because the relationship can be inferred from the *sentence-pattern* table and the *question-pattern* table as SELECT sentence FROM SENTENCE-PATTERN WHERE QUESTION-PATTERN.question=aquestion AND SENTENCE-PATTERN.pattern=QUESTION-PATTERN.pattern
3. **question** : What kinds of____ are there? What parts does ____ have? etc...
4. **pattern** : contains " called ", "named" or " means that ", etc...
5. **question-pattern** : A Many-to-Many relationship between question type and pattern type
6. **pattern-string** : A One-to-Many Unidirectional relationship between pattern type and pattern string

sentence-pattern		
id	**sentence**	**pattern**
0	345	34
1	345	67

sentence-question		
id	**sentence**	**question**
0	345	1
1	345	4
2	345	5

question			
id	**code**	**params**	**type**
0	1	1	What is the definition of * ?
1	2	1	What is a broader term(super) for * ?
2	3	1	What kinds of * are there?
3	4	1	What parts does * have?
4	5	1	What is * a part of?
5	6	2	How do I *1 a *2 ?
6	7	1	What is the function of *?
7	8	2	When does *1 *2 ? or When can I *1 *2?
8	9a	1	What happens if * ? or What does * cause?
9	9b	1	What causes * ?
10	10	1	What does * require/support?
11	11a	1	Where is * ?
12	11b	1	Where does * ?

pattern	
id	**words**
0	none
1	", that is ," or "; that is ,"
2	" called ", "named"
3	" means that "
4	" is a ", " are a ", " is an ", " are an ", " are ", "was a", "was an", "were a", "were an", "were"
5	"*efinition ", " defined as "
6	" are those ", " is the ", " are the ", " is any ", " are any ", "was the", "were the" ("was any", "were any", "were those" not useful)
7	" term ", "termed"
8	"known as"
9	"refer to" NOT USED
10	" includ*"

11	" types of ", " kinds of ", " categor*", "type of", "kind of" ,"a form of"
12	" such as "
13	" version of ", " versions of "
14	" and other ", " or other "
15	" other than "
16	" especially "
17	" like "
18	compound nouns
19	"contains", "contain","containing","contained","consists of", " consist of", "is made up of", "is composed of", "includes", "built", "is comprised of", "possess*","is contained by", "makes up", "made from","comprises", "is a constituant of", "divided into", "content*","ingredient*", "combin*", "form", "forms", "made up of"
20	"is an element of", "is a component of", "is a piece of", "is a member of", "part of"
21	possessive (adj) noun look for ^G(enitive) tag
22	"use", "using"
23	"you", "programmer", "user"
24	"know", "want", "learn", "like"
25	"how to"
26	"provid*", "offer*", "support*"
27	"uses","using", "used", "utilize", "utilized as", "uses to which","for use in","for using in"
28	"essential to","essential for","needed for"
29	"function"
30	"in order to" NOT USED
31	"serve* as"
32	"designed for"
33	"among" NOT USED
34	"combin*" NOW PART OF 19 August 5/98
35	"occur* in" NOT USED
36	"proportion","proportions
37	"*-rich", " rich* ... in ", " high* ... in ", " low* ... in "
38	"its"
39	"classified as"
40	"is one of", "are one of" ,", one of" , "is one"
41	"amount of", "amounts of"
42	"found in", "find in", "present in","appear in","appears in","appeared in"
43	"when", "while", "before", "after", "until", "time"
44	"if" - part of 48 now
45	infinitive
46	"on", "in" ,"between", "through", "from", "at", "where", "inside", "outside, "across", "into", "along"
47	support*, require*, need*, must, apply to, available for, is compatible with, maximum, minimum, only, allow, specify
48	result in, results in, result from, results from, result of, results of, cause*, induce*, create*, if, when, reason*
49	", particularly ", ", and particularly ", "- particularly "
50	"one of ... is ", "One of ... is " (beg. of sentence)
51	" involves "
52	", in which ", "- in which " and not "which case"
53	"occurs when "
54	"(...)"
55	":"
56	noun "in" noun
57	"concentration ... in"
58	"what is", "what are"
59	"as a", "as an"
60	"e.g", "eg"
61	", which is the", ", which is a"
62	"developed to", "developed for"

63	"performed to"
64	"by"
65	"allows"

The relationship between the question and pattern table is Many-to-Many Bi-directional, and is represented as :

question-pattern		
id	**question**	**pattern**
1	0	1
2	0	2
3	0	3
4	0	4
5	0	5
6	0	6
7	0	7
8	0	8
9	0	40
10	0	50
11	0	51
12	0	52
13	0	53
14	0	54
15	0	55
16	0	58
17	1	2
18	1	4
19	1	5
20	1	6
21	1	7
22	1	8
23	1	10
24	1	11
25	1	12
26	1	14
27	1	15
28	1	16
29	1	17
30	1	39
31	1	40
32	1	49
33	1	50
34	1	54
35	1	55
36	1	59
37	1	60
38	1	61
39	2	4
40	2	6
41	2	10
42	2	11
43	2	12
44	2	13
45	2	14
46	2	15
47	2	16

48	2	17
49	2	18
50	2	39
51	2	40
52	2	49
53	2	50
54	2	59
55	2	60
56	2	61
57	3	19
58	3	20
59	3	21
60	3	36
61	3	37
62	3	38
63	3	41
64	3	42
65	4	19
66	4	20
67	4	21
68	4	36
69	4	37
70	4	38
71	4	41
72	4	42
73	5	22
74	5	23
75	5	24
76	5	25
77	5	45
78	6	26
79	6	27
70	6	28
71	6	29
72	6	31
73	6	32
74	6	62
75	6	63
76	6	64
77	6	65
78	7	43
79	8	48
80	9	47
81	10	46

The pattern table can be further normalized into One-to-Many Unidirectional relationship between a pattern type and the word strings that match it. Quotes are not included in the table, they are used to highlight required spaces.

pattern-patstring		
id	**pattern**	**patstring**
0	0	none
1	1	", that is ,"
2	1	"; that is ,"
3	2	" called "
4	2	"named"

5	3	" means that "
6	4	" is a "
7	4	" are a "
8	4	" is an "
9	4	" are an "
10	4	" are "
11	4	"was a"
12	4	"was an"
13	4	"were a"
14	4	"were an"
15	4	"were"
16	5	"*efinition "
17	5	" defined as "
18	6	" are those "
19	6	" is the "
20	6	" are the "
21	6	" is any "
22	6	" are any "
23	6	"was the"
24	6	"were the"
25	7	" term "
26	7	"termed"
27	8	"known as"
28	9	"refer to"
29	10	" includ*"
30	11	" types of "
31	11	" kinds of "
32	11	" categor*"
33	11	"type of"
34	11	"kind of"
35	11	"a form of"
36	12	" such as "
37	13	" version of "
38	13	" versions of "
39	14	" and other "
40	14	" or other "
41	15	" other than "
42	16	" especially "
43	17	" like "
44	18	...
45	19	"contains"
46	19	"contain"
47	19	"containing"
48	19	"contained"
49	19	"consists of"
50	19	" consist of"
51	19	"is made up of"
52	19	"is composed of"
53	19	"includes"
54	19	"built"
55	19	"is comprised of"
56	19	"possess*"
57	19	"is contained by"
58	19	"makes up"

59	19	"made from"
60	19	"comprises"
61	19	"is a constituant of"
62	19	"divided into"
63	19	"content*"
64	19	"ingredient*"
65	19	"combin*"
66	19	"form"
67	19	"forms"
68	19	"made up of"
69	20	"is an element of"
70	20	"is a component of"
71	20	"is a piece of"
72	20	"is a member of"
73	20	"part of"
74	21	...
75	22	"use"
76	22	"using"
77	23	"you"
78	23	"programmer"
79	23	"user"
80	24	"know"
81	24	"want"
82	24	"learn"
83	24	"like"
84	25	"how to"
85	26	"provid*"
86	26	"offer*"
87	26	"support*"
88	27	"uses"
89	27	"using"
90	27	"used"
91	27	"utilize"
92	27	"utilized as"
93	27	"uses to which"
94	27	"for use in"
95	27	"for using in"
96	28	"essential to"
97	28	"essential for"
98	28	"needed for"
99	29	"function"
100	30	"in order to"
101	31	"serve* as"
102	32	"designed for"
103	33	"among"
104	34	"combin*"
105	35	"occur* in"
106	36	"proportion"
107	36	"proportions
108	37	"*-rich"
109	37	" rich* ... in "
110	37	" high* ... in "
111	37	" low* ... in "
112	38	"its"

113	39	"classified as"
114	40	"is one of"
115	40	"are one of"
116	40	", one of"
117	40	"is one"
118	41	"amount of"
119	41	"amounts of"
120	42	"found in"
121	42	"find in"
122	42	"present in"
123	42	"appear in"
124	42	"appears in"
125	42	"appeared in"
126	43	"when"
127	43	"while"
128	43	"before"
129	43	"after"
130	43	"until"
131	43	"time"
132	44	"if"
133	45	...
134	46	"on"
135	46	"in"
136	46	"between"
137	46	"through"
138	46	"from"
139	46	"at"
140	46	"where"
141	46	"inside"
142	46	"outside"
143	46	"across"
144	46	"into"
145	46	"along"
146	47	support*"
147	47	"require*"
148	47	"need*"
149	47	"must"
150	47	"apply to"
151	47	"available for"
152	47	"is compatible with"
153	47	"maximum"
154	47	"minimum"
155	47	"only"
156	47	"allow"
157	47	"specify"
158	48	"result in"
159	48	"results in"
160	48	"result from"
161	48	"results from"
162	48	"result of"
163	48	"results of"
164	48	"cause*"
165	48	"induce*"
166	48	"create*"

167	48	"if"
168	48	"when"
169	48	"reason*"
170	49	", particularly "
171	49	", and particularly "
172	49	"- particularly "
173	50	"one of ... is "
174	50	"One of ... is "
175	51	" involves "
176	52	", in which "
177	52	"- in which "
178	53	"occurs when "
179	54	"(...)"
180	55	":"
181	56	"in"
182	57	"concentration ... in"
183	58	"what is"
184	58	"what are"
185	59	"as a"
186	59	"as an"
187	60	"e.g"
188	60	"eg"
189	61	", which is the"
190	61	", which is a"
191	62	"developed to"
192	62	"developed for"
193	63	"performed to"
194	64	"by"
195	65	"allows"

Mapping to Enterprise JavaBeans

Using Enterprise JavaBeans, 8 of the normalized Text Analyzer tables can be represented as entity beans and 3 of the tables become container managed relation fields. CMP fields are container managed persistence fields and CMR fields are container managed relationship fields. An entity bean represents a row in a table. Some of the relationship tables are not directly represented as entity beans but are instead implemented using CMR fields. All primary key fields (id column) are container managed, i.e. they are implicitely handled by the container.

Here are the 8 tables which map directly to Entity beans :

High traffic tables : The following tables are updated with a very high frequency and also require the most resources. Care should be taken when deploying the enterprise application that these tables get mapped to a separate database tablespace on a separate disk partition or better still, on a dedicated high speed hard drive. The relationship tables which map to these Entities should also be considered high traffic.

1. **Sentence**
2. **StemmedWord**
3. **Occurrence**

Low traffic tables : Tables which are updated with a very low frequency. These very small tables should be placed in a tablespace separate from the high traffic tables. Better still, these tables should be permanently memory cached.

1. **SyntacticTag**
2. **PartOfSpeechTag**
3. **Question**
4. **Pattern**
5. **PatternString**

The relationships between tables are sometimes also represented as other tables, however when using Enterprise JavaBeans some of these tables are automatically managed by the EJB container. Here are the 3 tables which map to container managed relationship fields :

- **sentence-pattern** : The Many-to-Many bi-directional relationship between Sentence and Pattern is represented as two CMR fields; getSentences() in Pattern and getPatterns() in Sentence.
- **sentence-question** : The Many-to-Many bi-directionalrelationship between Sentence and Pattern types is represented as two CMR fields; getSentences() in Question and getQuestions() in Sentence.
- **question-pattern** : The Many-to-Many bi-directional relationship between Question and Pattern is represented as two CMR fields; getQuestions() in Patterns and getPatterns() in Question.

Pre-processor Information Loss

One issues that has to be resolved is the information loss when the pre-processor tags the *Sentence* with *Pattern* numbers. Information such as the position of the *PatternString* within the *Sentence* is critical during pattern refinement because it is required when determining that the proximity of the search word is within positional tolerance. We recommend including the pattern position, also we recommend storing a reference to the exact *PatternString* instead of the more general *Pattern*.

To implement this change, the **sentence-pattern** and **sentence-question** relationships can be replaced with a new Entity bean called **SentencePatternString** containing the position of a *PatternString* within the *Sentence*. This improves the precision of knowledge of the position and defines the exact *PatternString* found in the *Sentence* as opposed to just associating a series of *Pattern* references with the *Sentence*. To find a *Sentence* that matches the *Question* type and search string we can replace the pattern refinement procedure with a container managed persistence finder method. The proximity of the search string to the *PatternString* can be determined using the procedure :

- Select *Sentences* belonging to *Question* type
- Select *Occurrences* belonging to *Sentences*
- Select *SentencePatternStrings* belonging to *Sentences*
- *Occurrence* must match the searchString
- The *Pattern* of the *PatternString* of the *SentencePatternString* must belong to the *Question* type because not all *SentencePatternStrings* associated with a *Sentence* belong to the specified *Question*.
- *Occurrence* position must be within the given tolerance of *PatternString* position stored in *SentencePatternString*

This procedure can be encoded using following EJB-QL statement (portable Query language for Enterprise JavaBeans) : With the following three parameters :

abstract Collection findSentences(Question **question**, String **searchString**, int **tolerance**);

1. **question** : the *Question* type
2. **searchString** : the search string
3. **tolerance** : the maximum number of characters that search word can be located from the *PatternString*

```
SELECT OBJECT(s) FROM
  IN(Question.sentences) as s,
  IN(s.occurrences) as o
  IN(s.sentencePatternStrings) as sps,
WHERE
  Question = ?1 AND
  o.word = ?2 AND
  sps.patternString.pattern MEMBER OF Question.patterns AND
  ABS(o.position - sps.position) < ?3
```

Concordance

A concordance on a single word search string is efficiently performed using the following finder method :

abstract Collection findSentences(String **searchString**);

```
SELECT DISTINCT OBJECT(s) FROM
  Occurrence AS o,
  o.sentence AS s
WHERE
  o.word = ?1
```

Unfortunately, the *Occurrence* table only stores single words. This leads to efficiency problems when the search string is a compound term. For a two word compound term the search procedure becomes very inefficient and complicated :

abstract Collection findSentences(String **searchString1**, String **searchString2**);

```
SELECT DISTINCT OBJECT(s) FROM
```

```
  Sentence AS s,
  IN(s.occurrences) AS o1,
  IN(s.occurrences) AS o2
WHERE
  o1.word = ?1 AND
  o2.word = ?2 AND
  o1.position + LENGHT(o1.word) + 1 = o2.position
```

If a sentence had 10 words this query would require 100 comparisons. This could be cut in half by adding the following where clause : o1.position > o2.position. However, the procedure efficiency remain of order $O(n^2)$ where *n* is the number of words in the sentence. Not good. Another problem that might arise in certain scenarios is that more than one space might be between the members of the compound term. This could be fixed by adding more clauses. Further research into how commercial search engines deal with the problem of compound terms would be beneficial.

Source Code

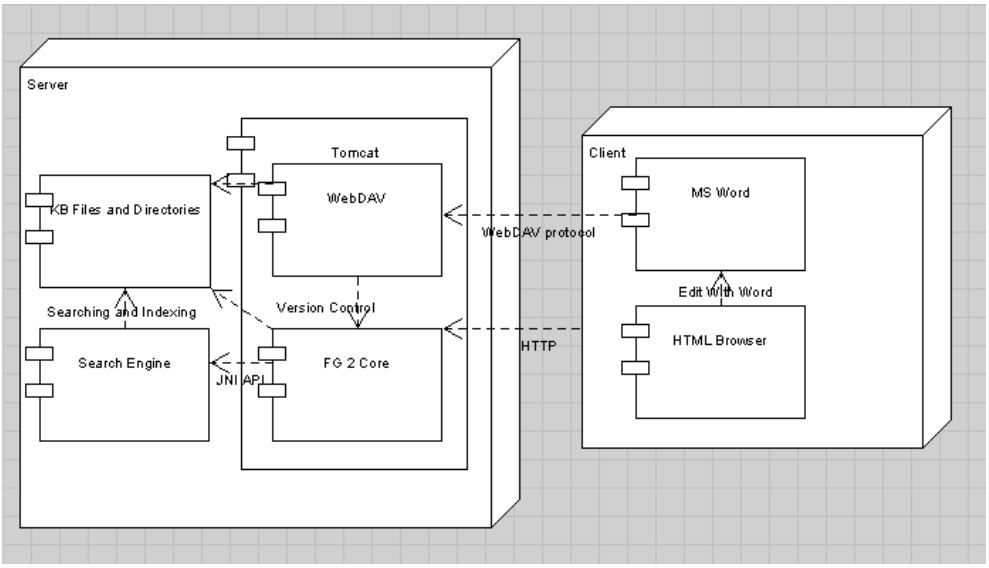
```
/** Java 2 Enterprise Edition version 1.3 representation of Text Analyzer data.
 * @author John Talbot
 */
public abstract class Sentence implements EntityBean {
  // Container managed persistence fields (CMP)
  public abstract void setDocument(int aDocument);
  public abstract int getDocument();

  public abstract void setSentenceNumber(int aSentenceNumber);
  public abstract int getSentenceNumber();

  public abstract void setParagraph(int aParagraph);
  public abstract int getParagraph();

  public abstract void setHeaderLevel(int aHeaderLevel);
  public abstract int getHeaderLevel();

  public abstract void setPreviousHeader(int aPreviousHeader);
  public abstract int getPreviousHeader();

  public abstract void setList(String aList);
  public abstract String getList();

  public abstract void setListHeader(int aListHeader);
  public abstract int getListHeader();

  /** The sentence content (the original sentence stripped of HTML). */
  public abstract void setContent(String aContent);
  public abstract String getContent();

  // CMR fields

  /** One-to-many, bi-directional relationship containing Occurrence entities */
  public abstract void setOccurrences(Collection someOccurrences);
  public abstract Collection getOccurrences();

  /** Many-to-many, bi-directional relationship containing Pattern entities */
  public abstract void setPatterns(Collection somePatterns);
  public abstract Collection getPatterns();

  /** Many-to-many, bi-directional relationship containing Question entities */
  public abstract void setQuestions(Collection someQuestions);
  public abstract Collection getQuestions();

  /** Many-to-many, bi-directional relationship containing SentencePatternString entities */
  public abstract void setSentencePatternStrings(Collection someSentencePatternStrings);
  public abstract Collection getSentencePatternStrings();


}

public abstract class StemmedWord implements EntityBean {
  // CMP fields
  /** Stemmed version of the word. */
  public abstract void setStem(String aStem);
  public abstract String getStem();
```

```
    // CMR fields
    /** one-to-many, bi-directional relationship containing Occurrence entities */
    public abstract void setOccurrences(Collection aSetOfOccurrences);
    public abstract Collection getOccurrences();
  }

  /** The Occurrence entity bean implements the many-to-many relationship
   * between the Sentence and StemmedWord entity beans.
   * Normally we would use a Many-to-Many CMR field in the Sentence EJB,
   * however we need to add extra information such as syntactic and
   * part of speech tags, position in sentence and word (non-stemmed version).
   * Therefore we create an explicit representation of the relationship.
   * This entity is cascade deleted when Sentence is deleted (is this possible?)
   */
  public abstract class Occurrence implements EntityBean {
    // CMP fields

    /** Position of the word in the sentence. */
    public abstract void setPosition(int aPosition);
    public abstract int getPosition();

    /** Unstemmed version of the word. For each stemmed word there exists one
     * or more unstemmed words which correspond to different forms of the word
     * such as singular/plural for nouns or past/present tense for verbs etc...
     */
    public abstract void setWord(String aWord);
    public abstract String getWord();

    // CMR fields

    /** Many-to-one, bi-directional relationship to a Sentence */
    public abstract void setSentence(Sentence aSentence);
    public abstract Sentence getSentence();

    /** Many-to-one, bi-directional relationship to a StemmedWord */
    public abstract void setStemmedWord(StemmedWord aStemmedWord);
    public abstract StemmedWord getStemmedWord();

    /** Many-to-one, bi-directional relationship to a SyntacticTag */
    public abstract void setSyntacticTag(SyntacticTag aSyntacticTag);
    public abstract SyntacticTag getSyntacticTag();

    /** Many-to-one, bi-directional relationship to a PartOfSpeechTag */
    public abstract void setPartOfSpeechTag(PartOfSpeechTag aPartOfSpeechTag);
    public abstract PartOfSpeechTag getPartOfSpeechTag();

  }

  public abstract class SyntacticTag implements EntityBean {
    // CMP fields
    /** A code used in external programs to mark the word as having the tag */
    public abstract void setCode(String aCode);
    public abstract String getCode();

    /** The meaning of the tag in natural language */
    public abstract void setTag(String aTag);
    public abstract String getTag();

    // CMR fields

    /** One-to-many, bi-directional container managed relationship field
     * containing Occurrence as collection members.
     */
    public abstract void setOccurrences(Collection aSetOfOccurrences);
    public abstract Collection getOccurrences();
  }

  public abstract class PartOfSpeechTag implements EntityBean {
    // CMP fields
    /** A code used in external programs to mark the word as having the tag */
    public abstract void setCode(String aCode);
    public abstract String getCode();

    /** The meaning of the tag in natural language */
    public abstract void setTag(String aTag);
    public abstract String getTag();
```

```
    // CMR fields

    /** One-to-many, bi-directional relationship to Occurrence entities. */
    public abstract void setOccurrences(Collection aSetOfOccurrences);
    public abstract Collection getOccurrences();
  }

  public abstract class Question implements EntityBean {
    // CMP fields
    /** A code used in external programs to mark a sentence as answering the question type */
    public abstract void setCode(String aCode);
    public abstract String getCode();

    /** The number of parameter that this question requires. */
    public abstract void setParameters(int aNumberOfParameters);
    public abstract int getParameters();

    /** The meaning of the question in natural language */
    public abstract void setType(String aType);
    public abstract String getType();

    // CMR fields
    /** One-to-many, bi-directional relationship to Sentence entities. */
    public abstract void setSentences(Collection aSetOfSentences);
    public abstract Collection getSentences();

    /** Many-to-one, bi-directional relationship to Pattern entities */
    public abstract void setPatterns(Collection aSetOfPatterns);
    public abstract Collection getPatterns();
  }

  public abstract class Pattern implements EntityBean {
    // CMP fields
    /** A comma separated set of PatternString */
    public abstract void setWords(String aPatternStringWords);
    public abstract String getWords();

    // CMR fields

    /** One-to-many, bi-directional relationship to Sentence entities. */
    public abstract void setSentences(Collection aSetOfSentences);
    public abstract Collection getSentences();

    /** Many-to-many, bi-directional relationship to Question entities */
    public abstract void setQuestions(Collection aSetOfQuestions);
    public abstract Collection getQuestions();

    /** One-to-many, bi-directional relationship to PatternString entities */
    public abstract void setPatternStrings(Collection aSetOfPatternStrings);
    public abstract Collection getPatternStrings();

  }

  public abstract class PatternString implements EntityBean {
    // CMP fields
    public abstract void setPatternString(String aPatternString);
    public abstract String getPatternString();

    // CMR fields

    /** One-to-many, bi-directional relationship to a Pattern. */
    public abstract void setPattern(Pattern aPattern);
    public abstract Pattern getPattern();

    /** One-to-many, bi-directional relationship to a SentencePatternString. */
    public abstract void setSentencePatternString(Collection aSentencePatternString);
    public abstract Collection getSentencePatternString();

  }

  public abstract class SentencePatternString implements EntityBean {
    // CMP fields
    /** Position of the PatternString within the sentence. */
    public abstract void setPosition(String aPosition);
    public abstract String getPosition();
```

```
    // CMR fields

    /** Many-to-One, bi-directional relationship to a PatternString. */
    public abstract void setPatternString(PatternString aPatternString);
    public abstract PatternString getPatternString();

    /** One-to-many, bi-directional relationship to a Sentence. */
    public abstract void setSentence(Sentence aSentence);
    public abstract Sentence getSentence();
}
```

Fact Guru 2 (FG2) Architecture



Fact Guru 2 Parts :

- Client Side
  - HTML Browser
  - Editor
  - File and Directory Manager
- Server Side
  - Application Server
  - Search Engine
  - ClearTalk Files and Directories

Client Side

- **Purpose** : To interact with Server side using a UI to view, author and manage ClearTalk files and directories.
- **Requirements**: Authorized users must be able to view, author and manage ClearTalk files from anywhere on the internet using a simple HTML editor/viewer.
- **Components**: An HTML browser and optionally an editor and file manager.
- **HTML browser**
  - **Function** : To view, navigate and query ClearTalk files and directories.
  - **Implementation**: IE5+, because it has WebDav capabilities and can launch MS Word.
  - **Compatibility** : Any windows computer for viewing and editing. However any computer connected to the internet can view and search an FG2 knowledge base.
  - **Interacts with** :
    1. MS Word to author a ClearTalk page by invoking the IE5 menu *File/Edit with Microsoft Word*
    2. Server Side by invoking Servlets and JavaServer pages using the HTTP GET protocol.
  - **Programming Environment** : HTTP/HTML/FORMS
  - **Current Status** : A demo search box is implemented using FORMS which invokes a Servlet running on the application server on the server side.
- **Editor**
  - **Function** : To author ClearTalk files.
  - **Implementation**: Microsoft Word 2002.
  - **Compatibility** : Any windows computer
  - **Programming Environment** : Visual Basic For Applications
  - **Interacts with** : WebDAV Server by allowing authors edit files on the Server Side
  - **Current Status** : Some VBA macros to convert a NL paragraph or a selection to a bulleted list of items enclosed in a tinted box just below the selection
- **File and Directory Manager**
  - **Function** : Manage ClearTalk files and directories on the Server side.
  - **Implementation**: Web Folders.
  - **Compatibility** : *Web Folders* are available on Windows XP or if IE5 and later is present. However there are also Linux WebDAV clients available.
  - **Programming Environment** : WebDAV Protocol
  - **Interacts with** : WebDAV Server by allowing authors edit files on the Server Side
  - **Current Status** : Works, however not password protected yet.
  - **Procedure** : To create a new Web Folder to our server on computer aif you must
    1. Open 'My Network Places'
    2. Under Network Tasks, click Add a network place.
    3. Click Next at 'Choose another network location'
    4. enter the IP http://www.site.uottawa.ca:1088/webdav

Server Side

- **Purpose** : To interact with Client Side and for storing and processing of ClearTalk files and directories.
- **Components**: An application server, ClearTalk files and a search engine.

- **Application Server**
  - **Function** : To interact with Client Side
  - **Implementation**: Tomcat 4.1.18 (Open Source)
  - **Compatibility** : Runs on Windows NT/2000/XP and Unix/Linux.
  - **Programming Environment** : Java Servlets and JavaServer Pages
  - **Interacts with** :
    1. HTML browser on Client Side for queries and browsing
    2. Editor on Client Side using WebDAV
    3. ClearTalk files for phrase extraction of Search Engine hits
    4. WebDAV Server by allowing authors edit files on the Server Side
    5. Search Engine to pass on queries on behalf of the Client Side
  - **Current Status** : Servlet for searching the three Java Documents at http://www.site.uottawa.ca:1088/search/index.html, the results page only returns one document with hits highlighted. TOMCAT is currently running as an NT/XP 'service' on two servers (i.e. you can log out of these machines but don't turn them off).
    1. aif (Doug's machine, Win XP) : http://www.site.uottawa.ca:1088
    2. aid (Slow lab computer, Win NT) : http://aid.site.uottawa.ca
  - **To do** : Phrase extractor to restrict search results to only the sentence, paragraph or ClearTalk statements containing the hits. Longer term goals include creating navigable concept hierarchies by extracting '*is a kind of*' facts. Topic hierarchies can be created by reflecting the directory structure in which the ClearTalk files reside.
- **Storage Manager**
  - **Function** : To allow remote users to read and write ClearTalk files and directories.
  - **Implementation**: WebDAV server within Tomcat. WebDAV is more secure than FTP.
  - **Compatibility** : Same as Tomcat.
  - **Programming Environment** : WebDAV Protocol
  - **Current Status** : Machine aif support a WebDAV folder at http://www.site.uottawa.ca:1088/webdav, however the current configuration has no password.
  - **To do** : The storage manager must inform the application manager of any changes in the ClearTalk files because the ClearTalk files need to be re-indexed by the search engine at some point (either immediately or periodically).
- **ClearTalk Files and Directories**
  - **Function** : To represent ClearTalk documents
  - **Implementation**: HTML files and directories which can represent topic hierarchies
  - **Compatibility** : Any computer
  - **Programming Environment** : ClearTalk, a kind of constrained natural language
  - **Interacts with** : Search engine, application server and storage manager
  - **Current Status** : We currently have 4 ClearTalk files (3 for Java, 1 for the ClearTalk syntax itself)
  - **To do** : Create a comprehensive set of ClearTalk files about something. For example the Java tutorial or some well known Java book.
- **Search Engine**
  - **Function** : To improve search efficiency by using index files
  - **Implementation**: dtSearch Engine v6.11 (Password DTENGINE6)
  - **Compatibility** : Runs on Windows NT/2000/XP only.
  - **Programming Environment** : Java using JNI
  - **Interacts with** :
    1. Application server using Java Native Interface (JNI)
    2. ClearTalk files by creating a proprietary index file to perform constant time searching.
  - **Current Status** : We currently have three java documents indexed, however the indexing operation must be performed while logged into the main server.
  - **To do** : Create an indexing Servlet which can index a set of ClearTalk files from a remote client or automatically when ClearTalk files are edited and uploaded via WebDAV.

Application Server - Design Decisions

An application server has two tiers :

1. **Web Services Tier**: Functional replacement for an Apache web server plus the computational power of Servlets and Java Server Pages. This first tier is where the HTML user interface resides. We are using Tomcat for this Tier.
2. **Enterprise JavaBeans Tier (EJB)**: A second tier containing a Java object based persistence engine backed by a relational database plus a galaxy of load balancing application server features ideal for heavy traffic web sites. The persistence mechanism provided by EJB is ideal for handling KB objects from FG1 type knowledge bases or wherever a database is called for.

For the prototype phase of FG2 we will not be using an EJB tier. however we should keep it in mind if we later consider a FG1 KB object cache for efficiency purposes. Most experienced J2EE programmers recommend starting with a web services tier and only later augmenting it with an EJB tier since this increases application complexity significantly. The web services tier can make use of several open source frameworks such as :

1. Java Standard Tag Library (JSTL) for JavaServer Pages
2. Jakarta Struts
3. Cocoon, etc...

These frameworks can greatly simplify the design of a web services user interface, however for the prototype will not be using them, except perhaps JSTL because it is now a Java standard.

Software Development Environment

1. ANT 1.5.2 - a very popular XML based build tool like make.
2. UltraEdit 10.0
3. Optionally NetBeans 3.4.1 or Sun One Studio 4 Community Edition
4. Windows XP Service Pack 1
5. Java 1.4.1_02

Implementation Details

Phrase Extraction

Part I : Fact Extraction

Two new ideas to both improve FG2 performance while dramatically simplifying the fact extractor code :

Method #1

Use the split(text) String method. Which takes a regular expression (regex) such as 'o-' and splits up the String at the match boundaries into an array of Strings.

String [] facts = text.split("o-");

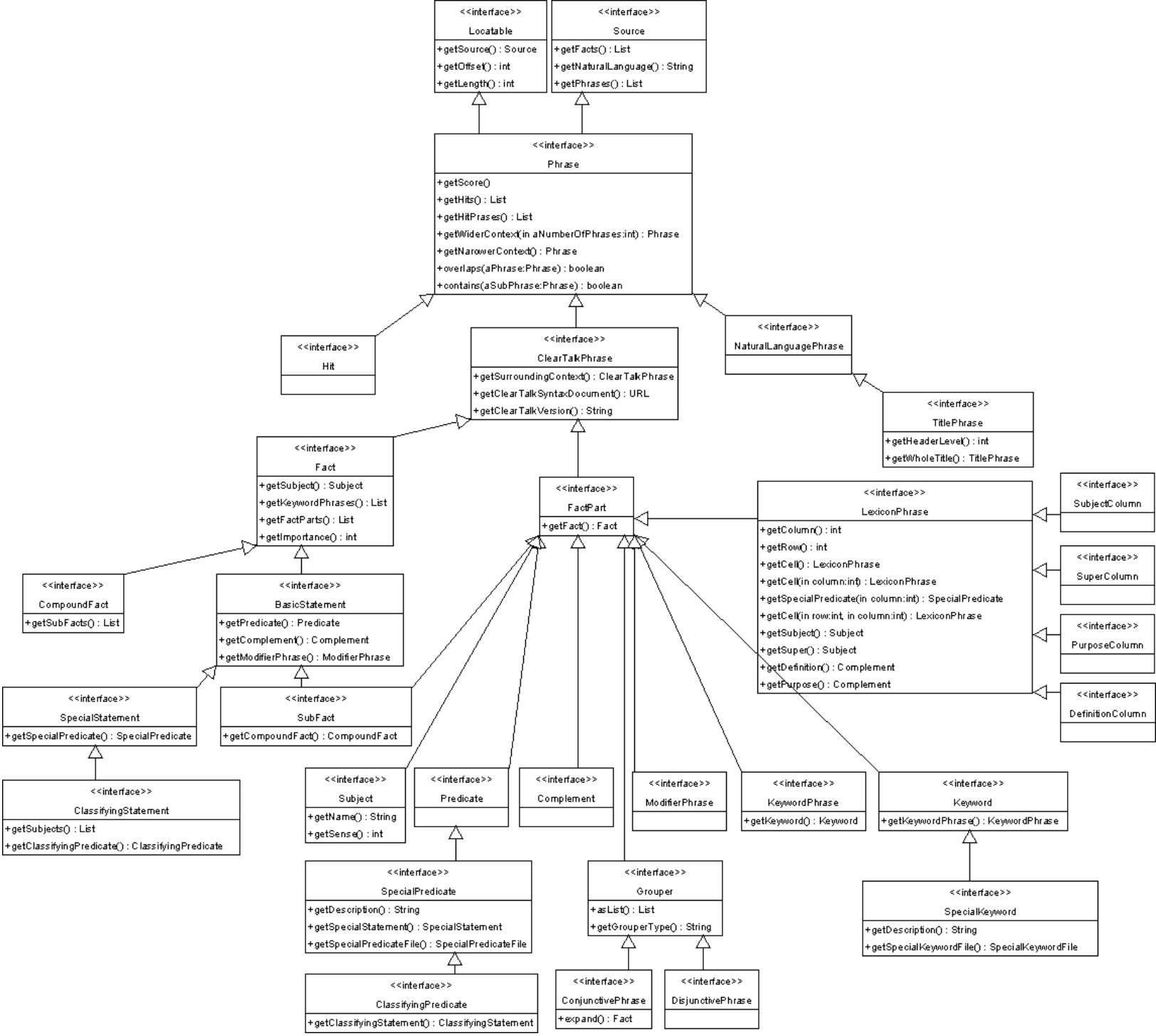
This method is syntactic sugar and actually delagates the invocation of split() to the Pattern regex class. Regex are new to Java 1.4. Now all that is left to do is find the entries in the facts array in which the hit(s) are located. There are several issues to consider :

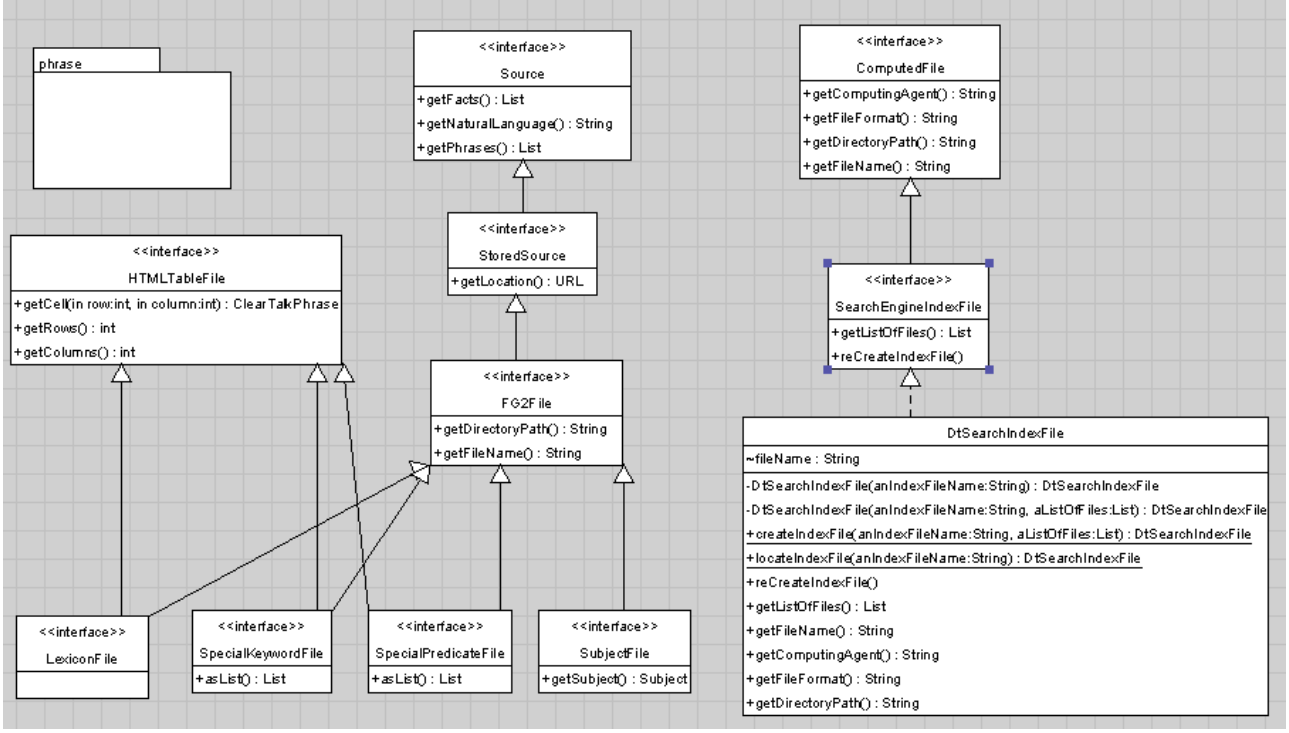
1. It is more robust than Martin's code when the hits are NL or other unusual circumstances.
2. If we choose the phrase to be the entire file, we could do all hits in one pass.
3. The technique has poor efficiency if the phrase is the entire file, so if it is too slow we should only split small phrases.
4. If the user requests a nearby context which is larger than the fact, this is easier to accomplish with split.
5. Naturally supports the expansion of compound or dash-dash facts, since we need to insert the subject into the beginning of each sub-fact of a dash-dash fact, we get this *'for free'* using split. All that's left to do is to locate the facts[] array element containing the subject and insert it at the beginning of all dependent sub-facts.

Method #2

1. We can achieve constant-time performance when splitting out CT and NL by using dtSearch to locate all the instances of the 'o-' string.
2. We can also search for the 'o-1' and 'o-2' end-of-fact markers to facilitate the separation of CT from NL
3. I just tried searching for 'o-' within dtSearch desktop on our three FG2 documents and it is 100 percent reliable in locating the beginning and end-of-fact markers.
4. Since the search is always the same, we can further improve performance by caching the locations of all the CT and of the NL that we discovered in steps 1 and 2. This caching can be done when the server starts up or when when re-indexing is required. This cache must be updated when a document changes. However this is a much lower priority TODO.

Class Diagrams


<<interface>>
Locatable
+getSource() : Source
+getOffset() : int
+getLength() : int
<<interface>>
Source
+getFacts() : List
+getNaturalLanguage() : String
+getPhrases() : List
<<interface>>
Phrase
+getScore()
+getHits() : List
+getHitPrases() : List
+getWiderContext(in aNumberOfPhrases:int) : Phrase
+getNarowerContext() : Phrase
+overlaps(aPhrase:Phrase) : boolean
+contains(aSubPhrase:Phrase) : boolean
<<interface>>
Hit
<<interface>>
ClearTalkPhrase
+getSurroundingContext() : ClearTalkPhrase
+getClearTalkSyntaxDocument() : URL
+getClearTalkVersion() : String
<<interface>>
NaturalLanguagePhrase
<<interface>>
TitlePhrase
+getHeaderLevel() : int
+getWholeTitle() : TitlePhrase
<<interface>>
Fact
+getSubject() : Subject
+getKeywordPhrases() : List
+getFactParts() : List
+getImportance() : int
<<interface>>
FactPart
+getFact() : Fact
<<interface>>
LexiconPhrase
+getColumn() : int
+getRow() : int
+getCell() : LexiconPhrase
+getCell(in column:int) : LexiconPhrase
+getSpecialPredicate(in column:int) : SpecialPredicate
+getCell(in row:int, in column:int) : LexiconPhrase
+getSubject() : Subject
+getSuper() : Subject
+getDefinition() : Complement
+getPurpose() : Complement
<<interface>>
SubjectColumn
<<interface>>
SuperColumn
<<interface>>
PurposeColumn
<<interface>>
DefinitionColumn
<<interface>>
CompoundFact
+getSubFacts() : List
<<interface>>
BasicStatement
+getPredicate() : Predicate
+getComplement() : Complement
+getModifierPhrase() : ModifierPhrase
<<interface>>
SpecialStatement
+getSpecialPredicate() : SpecialPredicate
<<interface>>
SubFact
+getCompoundFact() : CompoundFact
<<interface>>
ClassifyingStatement
+getSubjects() : List
+getClassifyingPredicate() : ClassifyingPredicate
<<interface>>
Subject
+getName() : String
+getSense() : int
<<interface>>
Predicate
<<interface>>
Complement
<<interface>>
ModifierPhrase
<<interface>>
KeywordPhrase
+getKeyword() : Keyword
<<interface>>
Keyword
+getKeywordPhrase() : KeywordPhrase
<<interface>>
SpecialPredicate
+getDescription() : String
+getSpecialStatement() : SpecialStatement
+getSpecialPredicateFile() : SpecialPredicateFile
<<interface>>
Grouper
+asList() : List
+getGrouperType() : String
<<interface>>
SpecialKeyword
+getDescription() : String
+getSpecialKeywordFile() : SpecialKeywordFile
<<interface>>
ClassifyingPredicate
+getClassifyingStatement() : ClassifyingStatement
<<interface>>
ConjunctivePhrase
+expand() : Fact
<<interface>>
DisjunctivePhrase



References

- How To Write ClearTalk Document, March 22, 2003
- Test Fact Guru 2 document with appearance controls
- JavaDoc API
- Source code ZIP file for the Model
- UML Class diagrams in Poseidon 1.6 format
- Special Predicates HTML File
- Keyword Phrases HTML File
- Old Fact Guru 1 Java knowledge base automatically converted to Fact Guru 2 format (723 subject files)

How to Write in ClearTalk

Doug Skuce

School of Information Technology and Engineering
University of Ottawa

March 22, 2003

1

2

3 Controlled Language and ClearTalk

ClearTalk is a kind of controlled language; it is also known as CT. comment: this document is written almost entirely in CT. | This document could be written entirely in CT.
Controlled language purpose: to make documents that are (very clear, concise and machine-processable). | The use of controlled language is a kind of skill that many people should learn; it could be taught in high school. | CT is a kind of methodology for clear writing; it helps you to write with mathematical precision if you are careful. | CT is indicated by Arial font in this document and ordinary language is indicated by Times New Roman in this document.

ClearTalk documents – (
- – can be converted to KR formalisms almost automatically. comment: If CT is written with fully specified syntax, which we normally do not do, conversion could be done entirely automatically.
- – can be translated to other natural languages almost automatically. comment: nearly all of the problems that plague automatic translation are avoided in CT.
- – usually contain (both CT and ordinary language).
- – are normally authored in a (word processor or HTML editor).
- – are HTML documents.
- – are usually accessed as Web documents via (both direct access and a special search engine).

).

4 CT statements

CT statements – (
- -- are written with (a specified font and certain delimiters). reason: to distinguish them from ordinary language (aka: natural language). comment: the default font is Arial.
- -- use a small set of keywords.
- -- have a very simple syntax that can be unambiguous. comment: If CT is to be processed totally automatically, eg for automatic inferencing, additional syntax information is required to make it unambiguous. This is undesirable in most cases where humans will do the processing since it would add an unnecessary layer of complexity. Examples of how this is done are given in the Example section.).

Comments may be written in (both natural language and CT). | Natural language means: unconstrained ordinary language. | The only syntactic ambiguities in CT statements are the in the attachment of post-modifying phrases (eg: prepositional attachment). | Lexical ambiguities can be removed by using (namespaces and sense numbers). | Any ambiguity would require manual intervention when translating into a formalism. | If the author desires, all syntactic information can be specified. purpose: to make the syntax unambiguous. comment: requiring this would make CT too formal; this much detail is not needed for most applications.

4.1 CT Statement syntax

CT statement syntax: basic statement || compound statement || special statement. comment: || means: or

Basic statement syntax: subject [predicate] [complement] [modifier phrases] [. keyword phrases]. ('.' newline || '.|' || '.' whitespace '|')
comment: a period is required before a keyword phrase if a statement has a predicate. purpose: better readability
comment: the statement terminator is a period followed by a (newline or vertical bar)
comment: the modifier phrases modify a preceding (verb or noun), but they can be ambiguous without syntactic information.
comment: keyword phrases purpose: to supply information in certain well-known semantic relationships.
comment: some keyword phrases may be in natural language.

Compound statements are discussed in the section (= Special Statements). | Special statements are discussed in the section (= Compound Statements). | A predicate is only omitted with certain keyword phrases.

4.1.1 Noun phrases

a noun phrase –(
-- is a kind of (proper noun, count noun, mass noun, or pronoun).
-- can be modified by adjectives unless it is a proper noun.
-- is (either singular or plural) if it is a count noun.
-- can be modified by quantifiers unless it is a proper noun. examples: the boy, 3 balls, few women.
-- can be modified by 1 or more prepositional phrases. examples: the cat on the mat near the door.
comment: here is an example of prepositional phrase ambiguity: does "near the door" modify "cat" or "mat"? We can clarify this by showing the syntax thus to have 'near the door' modify 'mat': (the cat (on the mat (near the door)).
-- can be modified by 1 relative clause. examples: (the man who came today; the man whose shirt is white; the box that is on the table)
-- can be modified by 1 qualifying phrase. example: ClearTalk (aka: CT) -1
). comment: the qualifying phrase must follow immediately.

The quantifier (= 'a', 'any', 'each' 'every' or 'all') on a subject means: universal.
The quantifier (= 'any', 'each' 'every' or 'all') on a non-subject means: universal.
non-subject means: a noun phrase that is not the subject.
The quantifier 'a' on a non-subject means: 1.
If a quantifier is not universal then it is existentially dependent on the subject and converselyo-1
The following are instances of quantifiers: no, some, many, a few, 0 or more (aka: 0m), 1m, most, less than <int>, from <int> to <int>, <int>. comment: any others that behave like these
A qualifying phrase = (one of: 'aka:', 'eg:', 'ie':, 'a:', 'called:', '=' a (noun or variable), a variable) examples: Microsoft (a: company), ClearTalk (aka: CT).
aka means: also known as. purpose: to introduce a synonym.
Every subject is an instance of a noun phrase.

4.1.2 Pronouns and references

A pronoun may be used later in a statement to refer to another noun in the statement if the referent that it refers to can be determined using simple rules. examples: (Jane called her friend; skiing is fun but it can be dangerous; John brought some boxes but they were too small).

('it' and 'itself' and 'they' and 'themselves') always refer to the subject.

To refer to some noun other than the subject you may use (either a pronoun or ('the' or 'that') followed by the noun). example: A variable that refers to an object contains a pointer to that object.

4.1.2.1 Variables

A variable may be (introduced in a qualifying phrase and later in place of a pronoun). example all students who are enrolled in a course (C) must have the prerequisite for C.

4.1.3 The possessive

You may use a possessive instead of a prepositional phrase with 'of' example: 'John's mother' instead of 'the mother of John'.

4.1.4 Lists

A list is a kind of sequence of two or more words that do not behave as an "and". example: (Bill, Joe, Bob) lifted the heavy table. | A list is enclosed in parentheses. | The elements of a list must be noun phrases and they can be separated by (commas, dashes, semicolons, or bullets). comment: (spaces and newlines) do not count as list punctuation. | A list with 'and' before the last element is not allowed. reason: it would be confused with a logical 'and'.

4.1.5 Relative clauses

Relative clauses –(
-- begin with ('that', 'which', 'who', 'whose' or 'what').
-- modify the noun that they follow.
-- contain a predicate.
-- may contain (a complement and modifiers).
).
Relative clause examples: (
The type of a variable determines which kinds of objects the variable may contain;
John knows the man who came here yesterday;
The person whose car is outside must move the car).

4.1.6 Embedded statements.

Sometimes a statement must be part of a larger statement, for example, when using verbs referring to mental acts. Such statements must be enclosed in double pairs of parentheses, brackets, or braces.

An embedded statement –(
– is part of a statement
– is delimited by a pair of (parentheses, brackets, or braces)
– example: Mary hopes that [[Bill loves her]]).
■

4.2 Conjunctive phrases

A conjunctive phrase –(
-- conjoins (nouns, verbs, adjectives or adverbs).
-- requires parentheses. example: (both the boy and the girl).
-- may contain a conjunctive phrase. example: (both Bill, (either Mary or Jane))
-- examples: (all of: red, green, blue), (red but not pink)
-- comment: note that conjunctive can mean OR.
)

The following are instances of conjunctive phrase connectors: (
none of
1 of
1m of 1 or more of
both – and –
some of
all of
but not
or
either – or -)

A conjunctive phrase is equivalent to two or more statements. example: Bob likes (both Mary and Jane) means: Bob

likes Mary and Bob likes Jane
).

Parentheses in conjunctive phrases. purpose: to eliminate syntactic ambiguity.

4.2.1 Predicates

A predicate --(
-- immediately follows the subject.
-- contains a verb phrase with 0m adverb modifiers. example: usually does not eat quickly.
-- may contain a noun-preposition pair that names a relationship if the verb is ('to be' or 'has'). examples: is the father of, is equivalent to, has as color.
-- may combine a passive verb with a preposition. example: is connected to.
).

4.2.2 Verb phrases

A verb phrase –(
-- must contain at least one verb
-- can contain a particle. examples: look up, skip over
-- can contain a conjunction of verbs. example: (eats and drinks)
-- can contain auxiliary verbs. example: (was going; has been eating)
-- can contain (modals and negation). example: must not go
-- can contain adverbs. example: (eats quickly; frequently stops)
).

4.2.3 Complements

A complement --(
-- immediately follows the predicate.
-- is usually a noun phrase. example: Bob loves Mary.
-- can be an infinitive phrase. example: Bob wants to marry Mary.
-- can be an adjective that modifies the subject if the verb = 'to be'. example: Mary is pretty.
).

If a complement is a instance of an infinitive phrase then it may have verb modifier phrases. example: Bill wants to quickly hold up the bank.

4.2.4 Modifier phrases

Modifier phrases –(
-- follow the (noun or verb) that they modify. example: Bill bought a car while shopping. comment: 'while shopping' modifies 'bought'.
-- are usually prepositional phrases.
-- begin with a word that acts like a (preposition or verb-modifying adverb or a conceptual relation).
-- are not explicitly delimited.
-- are subject to possible syntactic ambiguity because they may not follow immediately.
).

Modifier phrases *could* be delimited to make them syntactically unambiguous but that is not something most people would want because it would be messy to read and write. The main situation where delimiting is important is when some automatic system is trying to process CT. If this were the case a human would define the disambiguation. To force a modifier phrase to modify a noun or verb that does not immediately precede it, use parentheses (see example section below.)

4.2.4.1 Specifying conceptual relations

If you wish to explicitly specify binary conceptual relations (eg agent, theme, source) then you can use the dash notation in a manner that is similar to conceptual graphs with the predicate as the initial part. example: go – (agent:

John) (dest: Boston) (instr: a bus). comment: you may want to compare this with the standard linear cg form.

4.2.5 Keyword phrases

The final component of a statement may be 1 or more keyword phases. | A keyword phrase begins with a keyword that ends in a colon. purpose: to add information to a statement. comment: keywords can be written with (a possessive subject and a verb) to make them more readable. example: a lock's purpose is: to prevent something opening.

Each keyword has a domain-independent semantics.

keywords instances: (
difference: purpose: states a difference between two concepts;
unlike: purpose: names a concept that is similar but different in some way, preferably stating the difference;
resembles: purpose: to state that something is similar;
reason/because: purpose: gives a reason;
purpose: purpose: gives a purpose;
causes/caused by: purpose: gives a causal relation
except: purpose: lists exceptions;
definition: purpose: gives the definition;
example: purpose: gives examples;
means: purpose: says what some phrase means;
parts: purpose: lists parts;
location: purpose: states a location
time: purpose: states a time
kinds: purpose: lists kinds (ie subtypes);
instances: purpose: lists instances;
syntax: purpose: gives syntax for language constructs;
code: purpose: to include code examples
where: purpose: to define words or symbols used previously
how: purpose: to state how to do something
why: purpose: to state a reason.
comment: to be followed by an infinitive
when: purpose: to state a time
followed by: purpose: indicates a syntactic sequence
). comment: many more could be imagined but these have sufficed for thousands of statements

4.2.6 Adverbs

Adverbs can only modify (verbs and adjectives). Any adverb that modifies a verb must be adjacent to (that verb or another adverb).

4.2.7 The pronoun 'you'

If a statement has a subject (= 'you') then it's purpose is to tell you (what you may do or how you may do something). To say how to do something you use a special statement that has (1m of: an infinitive, 'you') example: to bake a cake you follow a recipe.

4.2.8 Use of period

A period must (precede every keyword phrase and terminate every statement). comment: you can tell the difference because a statement cannot begin with a keyword and colon.

4.2.9 Use of bold, italics, underlines and fonts.

(Bold, italic and underline) have no significance in CT; you may use them as you wish.

(You must write all your CT statements in the same font within any document; this font must be different from any

other font in that document) or you must put delimiters on CT statements.

4.2.10 Statement delimiters and levels

If you have put delimiters on a CT statement then the statement does not need a special font. | CT is distinguished from natural language by (either a font or delimiters or both). comment: this document uses both. | Delimiters are easier for automatic processing but do not look as nice as special fonts. o-<n>') may be placed at the end of a CT statement. where: n represents a digit indicating the importance level. if n=1 then this statement has prime importance. means: the statement should be learned before ones of lesser importance. | Importance levels purpose: to permit summarization. how: by showing only statements above a certain level.

4.3 Special Statements

Special statements are statements that have certain special predicates that serve a fixed semantic purpose.

The following are instances of special predicates: (

- is a kind of
- is an instance of. comment: the phrase "is a" is never used without "kind" or "instance" purpose: to avoid ambiguity.
- is different from
- is similar to
- is the same as | is equal to | =
- is part of
- has as part || has as parts
- is disjoint from || is distinct from
- is also known as || is a synonym of

) comment : the plural form of each of the above is permitted; the negated form of each is permitted.

There are obviously more possible special predicates, though not a lot more. These are the ones that have proved useful so far.

4.3.1 Classifying Statements

There are 4 kinds of classifying statement: –(

-- the *kinds-of* classifying statement.
-- the *instances-of* classifying statement.
-- the *following-are-kinds -of* classifying statement.
-- the *following-are-instances-of* classifying statement.
).

kinds-of classifying statement. syntax: There are <n> kinds of *noun phrase* [based on *noun phrase*]: (*list of noun phrases*). means: these are a disjoint partition based on some criterion. comment: the optional 'based on' phrase can name the attribute that causes the distinction. example: there are 4 kinds of people based on age: (infants, children, teens, adults).

instances-of classifying statement. syntax: There are <n> instances of *noun phrase:* means: these are all the instances there are. example: there are 10 instances of digits: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

following-are-kinds-of classifying statement. syntax: the following are kinds of *noun phrase:* (*list of noun phrases*). purpose: to list kinds of something. means: these are some of the kinds; there may be others. example: the following are kinds of mammal: (bear, deer, pig, cow).

following-are-instances-of classifying statement. syntax: the following are instances of *noun phrase.* purpose: to list instances of something means: these are some of the instances; there may be others.

4.4 Compound Statements

a compound statement --(
– may be constructed using 1m statement combiners.

– may be constructed using the dash-dash construction for multiple statements that have the same subject. example: this statement uses the dash-dash construction. comment: see below.
).

4.4.1 Statement combiners

The following are instances of statement combiners: (
;
and
or
but
but not
if
if -– then
if and only if (aka: iff)
if -– then – and conversely/vice-versa means: iff
unless
because
whenever
when
while
). comment: any word that behaves as a statement combiner and has a clear meaning can be used. | comment: if more than one combiner is used, parentheses must be used to show associativity. example: if (X knows Y and Y knows Z) then X can meet Z

4.4.2 The dash-dash construction

The dash-dash construction is a convenient syntactic sugar for multiple statements that have the same initial part; it is expanded by replacing each – in a statement by the global initial part from the first line. In this document, the initial parts are always just the subject, but sometimes the subject and predicate are the initial part.

The initial part of the dash-dash construction is, usually just the subject; it may include the predicate. example:
my cat ate – (
– 1 mouse,
– at least 3 rats,
– my homework).

Many examples of the dash-dash construction have appeared in this document.

4.4.3 Temporal keywords

When a temporal sequence is implied, you must use a temporal keyword to connect two statements.

The following are instances of a temporal keyword: (

- before
- after
- when
- while
- whenever
- and then
- next,
) example: Bob bought a car after he got a job.

4.4.4 Procedural knowledge

Procedural knowledge should be expressed in a subset of C. comment: we have not bothered to define a minimal “beginners” subset of C.

4.4.5 Use of paragraphs and new lines in Word:

Word has two kinds of new lines: *enter*, and *shift-enter*. The former produces a paragraph mark, that can contain a great deal of formatting. The latter merely forces new line. You may put either in CT statements to improve readability and they have no meaning. In either case, if a statement is ending, there must be a period.

5 Terms and the Lexicon

A term is a word that acts as a unit of referral. It can be made up of several words. All terms are entered in the lexicon and given a definition or purpose, and hopefully a genus (super). It is up to the author to enter terms in the lexicon. An authoring system would be of great help here. If a word is not in the lexicon, it is not a term, but maybe a potential term. If a term has several senses, they are distinguished by appending sense numbers after a dash. For work on the semantic web, lexicons would be designated by urls. Note: all of this paragraph can be written in CT.

sense number example: bank-1, bank-2. comment: this would distinguish two meanings of bank. comment: as a standard, one could use the Wordnet sense numbers.

Most (nouns and complements) are terms. reason: they are important.

All terms must be (entered in the lexicon and given (a definition or purpose)) comment: many definitions involve a purpose; if a definition involves a purpose you should use the keyword 'purpose' instead of: 'definition'. examples: hammer. purpose: a tool that is used to hit nails; triangle definition: a polygon with three sides.

All terms should ideally have a genus (aka: super). reason: so they may be put in a hierarchy..

Many terms are compound (aka: multi-word). example: car door mounting bolt.

The lexicon is normally maintained as a table in our system. comment: this makes it easy to export to various formats. comment: the lexicon can obviously be maintained in any representation.

6 Examples

This section of this document provides many examples. purpose: to assist in better understanding CT structures. Each example will be parsed using parentheses. purpose: to illustrate what the explicit syntax of CT is like if it must be specified. We also illustrate mixing unconstrained language (in Times New Roman) with CT.

> s will denote the subject.
> p will denote the predicate.
> v will denote the verb.
> c will denote the complement.
> m will denote modifiers.

If a modifier is not attached by parentheses to the preceding noun phrase then it modifies the predicate. If a modifier begins with 'of' then it modifies the preceding noun.

(s: The cat) ((p: ate) (c: the fish) (m: on the mat)).

> We have shown one possible parse. The other is where m: modifies 'fish': ((c: the fish) (m: on the mat)), which is the default if the parentheses are not given. This is a typical example of prepositional attachment ambiguity.

(s: Bill) ((p: usually dates) (c: Mary) (m: on weekends.) (because: (s: he) (v: is) (c: busy)).

(s: Java) ((p: was developed) (m: at Sun Microsystems) (m: in the mid-nineties)).

Note that 'in the mid-nineties' is not a location but a time period and hence could not modify Sun Microsystems but most parsers are not that smart.

(s: All Java compilers) ((p: compile) (c: source code) (m: into files (m: that (v: contain) (c: bytecode)))).

This example shows how even a simple syntactic structure can be messy to specify formally.

o-((s: A Java VM) (p: will not allow) (c: violations (m: of certain security constraints))) when ((s: applets) (p: are downloaded) (m: over the Internet.)).

The keyword is 'when' connects two statements.

((s: You) (p: do not need) (c: to free (c: objects) (m: from memory))) when ((s: you) (p: no longer need) (c: these objects). (unlike: (C, C++))o-1
o-(s: Java) (p: is less efficient than) (c: (C or C++)) because: (
(s: Java's (safety checks and garbage collection)) slow down execution) and
(((s: the interpretation) (m: of its bytecode)) (p: is not as fast as) (c: direct execution of machine code)).

Note that the parentheses around 'safety checks' and 'garbage collection' are always included in the unparsed CT (enclosing any conjunctive phrase) to avoid serious syntactic ambiguity. To expand the 'or', write the statement twice, once with 'C' and once with 'C++' and connect them with 'or'

6.1 The verb "to be"

The verb "to be" behaves specially. Any noun , preposition or adjective that directly follows it is considered part of the predicate. However the verb "to be" also occurs in some of the special statements, where its behaviour is more specific. (See Special Statements)

(s: Java) (p: is popular).

(s: Source code) (p: is in) (c: files).

(s: John) (p: is the father of) (c: Mary).

(s: Java) (p: is easy) (c: to use (m: for networking)).

6.2 "How to" with 'you' and 'how:'

(s: You) (p: define) (c: inheritance) (m: by creating (c: an inheritance hierarchy)).

This example illustrates using the pronoun 'you' to give "how to" instructions. The basic pattern is 'you' verb1 modifiers1 'by' 'verb2 modifier2'. An equivalent form is 'To' verb2 modifier2 'you' verb1 modifiers1. The keyword 'how:' could be inserted after 'inheritance' to make this sentence easier to identify as a "how to" fact.

An *informal procedure* involves giving "you" a number of steps, as in the steps for baking a cake:

To bake a cake --(
→ you get the ingredients.
→ and then you mix the ingredients together in a bowl to make the batter.

→ and then you dump the batter into a pan.
→ and then you bake the batter in the pan in an oven for several hours).

The keyword 'how:',which has default subject 'you', provides a third variation on how to write "how to" statements: Note the use of arrows instead of – to indicate that the order is significant.

A class is defined how: by writing a class declaration.

If there is an example then a 'you' statement may be omitted in a how-to statement:

To access an instance variable of an object (O) example: aVariable = O.variableName.

You may use an infinitive, as shown above.

6.3 Compound statements

The following illustrates a 'how-to' statement with a compound using if-then and the *followed by* keyword to indicate a syntactic sequence, which is a special kind of list.

If the code is in another class
thenyou access a class variable by writing (
the name of the class,
followed by: a dot,
followed by: the name of the variable) example: Student,numberOfStudents.

6.4 Syntax and code examples:

(s: class) (syntax: 'class' *className classBodyBlock*).

We permit statements without verbs for some keywords.

(s: 'syntax' (a keyword)) (p: is used for) (c: BNF definitions (m: of syntactic structures)).

(s: syntax (m: for BNF)) (p: is not specified). comment: we use italics for nonterminals and plain font for terminals.

The following illustrates how to specify syntax and give a code example. The use of a different font for the code is suggested:

method syntax: access modifier return type name (arguments) body where:
access modifer is an instance of an access modifier
return type is an instance of a (type or class) name
arguments is an instance of a list of zero or more argument declarations
body is an instance of a block.

method1 (a method) code:

```
public double credit(double amountToCredit)
{
    balance = balance + amountToCredit;
    return balance;
} .
```

The following preferences control the appearance of Fact Guru 2 documents :

- ☐ Show Clear Talk facts
- ☐ Show subject hyperlinks
- ☐ Bullet and highlight facts (as opposed to bar '|' delimited facts)

The JavaScript code is platform-independent and in a separate file. The stylesheet properties are also in a separate file and define the appearance of custom semantic HTML classes which mirror the objects present in a ClearTalk document such as subject, predicate, special-predicate, keyword, fact, facts (group of facts) or natural-language. This paradigm aids in separating content from presentation (as in XML versus XSLT/XHTML) The response is rapid because the changes occurs only on the client-side (reducing network traffic and load on our server). **Note:** The green boxes and bullets are not 'true' HTML constructs, they are style sheet manipulations of the P tag with 'facts' class and the SPAN tag with 'fact' class. (Same idea for subjects, predicates, complements, keywords etc...)

The Basics of Java

Java is an object-oriented programming language and, like *C++*, shares much of its syntax with the C programming language. If you know how to write statements, including variable declarations and loops, in one of these three languages, then you have a head start in learning the others. However, many other details differ among the languages. The biggest difference is that C is not object-oriented while the other two are. Also, the data types and libraries available are considerably different.

- Java **is a kind of** object-oriented programming language
- Java *is similar to* (C++'s and C's) syntax
- (Java and C++) *are* object oriented
- Java difference: C *is not* object-oriented

Java was developed at Sun Microsystems in the mid-nineties. It has a useful combination of features that, combined with the fact that it is a member of the C/*C++* family, have made it very popular in recent years.

- Java *was developed by* Sun Microsystems in the mid-nineties
- Java *is* very popular

Platform independence

Java is designed to be run using a *virtual machine* , or *VM*. Java compilers compile source code (typically found in files ending with the .java suffix) into files containing *bytecode* (typically in files ending with the .class suffix, or in libraries ending with *.jar)*. Java source code is also found in JServer pages which are files which end with .jsp. These pages are JavaSoft's version of MicroSoft's Active Server Pages.

- Java *runs using* a virtual machine
- Java compilers *compile* source code into files containing bytecode
- source code *is typically* (in files whose names end with .class or in libraries whose names

end with .jar)
- Java source code *is also found in* Java Server Pages which are files whose names end with .jsp
- Java Server Pages *are* Javasoft's version of MicroSoft's Active Server Pages

Bytecode is like a universal machine language – it is very low-level in the sense that it is not designed to be read by human beings. At the same time, programs in bytecode can be run on any computer that has a VM. The VM acts as an *interpreter* for the bytecode; this makes Java programs portable. Bytecode compiled on one computer can be run on a different architecture. The VM itself is a complex program usually written in C or *C++*.

- Bytecode *is like* a universal machine language; it is very low-level
- Low-level means: not designed to be read by human beings
- bytecode *can be run on* any computer that has a VM
- a VM *acts as* an interpreter for bytecode
- Java programs *are* portable because they are written in bytecode
- a VM **is a kind of** program that is usually written in C or *C++*

TODO: The rest of the Java document has not been converted to this new format. This will soon be automatic given any document following the well defined FG2 grammar.

Special Predicates

The plural form of special predicates is permitted; the negated form is also permitted.

Special Predicate	has purpose
is a kind of	to avoid ambiguity, the phrase 'is a' is never used without 'kind' or 'instance'
is an instance of	to avoid ambiguity, the phrase 'is a' is never used without 'kind' or 'instance'
is different from	diffenrentiator
is similar to	similarity
is the same as	equality
is equal to	equality
=	equality
is part of	meronomic relationship
has as part	meronomic relationship
has as parts	meronomic relationship
is disjoint from	subclasses
is distinct from	disctinction
is also known as	acronym
is a synonym of	synonymy
kinds of	Used in classifying statement representing a disjoint partition based on some criterion
instances of	Used in classifying statement representing an enumeration of instances
following are kinds of	Used in classifying statement to list some of the kinds of something
following are instances of	Used in classifying statement to list some of the instances of something

Keyword Phrases

keyword	has purpose
aka	also known as
because	gives a reason
caused by	gives a causal relation
causes	gives a causal relation
code	to include software code examples
comment	comment
definition	gives the definition
difference	states a difference between two concepts
eg	-
example	gives examples
except	lists exceptions
followed by	indicates a syntactic sequence
how	to state how to do something
instances	lists instances
kinds	lists kinds (ie subtypes)
location	states a location
means	says what some phrase means
one of	exclusive or
1m of	logical or
parts	lists parts
purpose	gives a purpose
reason	gives a reason
resembles	to state that something is similar
syntax	gives syntax for language constructs
time	states a time
unlike	names a concept that is similar but different in some way, preferably stating the difference
when	to state a time
where	to define words or symbols used previously
why	to state a reason

Drupal FactGuru

2011-05-11

To facilitate the migration towards JFG and DFG I establish the requirement that a minimum number of changes be made to the existing system. Drupal requires MySQL so does the old FG so for now we stick to MySQL. The latest Java Developers Kit has a built-in database engine : Java DB is Sun's supported distribution of the open source Apache Derby written in Java. I will FG also working on Java DB. Derby was not available when FG was developed but now offers us the capability to run an RDBMS entirely from memory speeding up the system by many orders of magniture.

As the system evolves and becomes more modular, I will convert parts of it to support Erlang and eventualy CouchDB. This is the philosophy of extreme programming, at every stage of the transformation, the system must pass a series of test designed to prove that it remains in a usable state. The advantages of the XP philosophy are too numerous to mention here but you can read about it in any XP book or ask Tim. I offer the following migration route between the old FactGuru (FG) and the new Java FactGuru (JFG) and Drupal FactGuru (DFG). These are two main areas of concern (A) Persistence and (B) Graphical User Interface (GUI).

(A) Persistence

Because the designers of FG had the foresight to use a DB for persistence, Drupal and FG can inter-operate via the DB layer. I propose two new environments from which a KB can be edited (1) JavaFG (JFG) which will allow people not interested in a CMS to use FactGuru using a Java tool and (2) Drupal FG (DFG) :

- **JFG** : I propose the use of the Java Persistence Architecture (JPA), a technology recently added to the core of Java 6. JPA is a fantastic enterprise quality object oriented persistence mechanism that is completely database independent, i.e. objects can be persisted to XML, a database or any other user defined format. JPA is also the main mechanism used in the Java 2 Enterprise Edition (J2EE) platform, the most popular application server on the web. JPA enhances the interoperability between FG and thousands of Enterprise applications. The current FG persistence engine is currently bloated with obsolete low-level procedural JDBC database handling code, this will be replaced with simple JPA. Also, Wratko created a highly complex caching mechanism to improve the efficiency of remote access over the Internet, however it's nearly impossible to figure out and it's buggy. Thankfully this will also be replaced with JPAs more efficient and stable enterprise quality caching mechanism which requires no maintenance because it is automatic/transparent/hidden by default. The JPA is standards-based and will reduce code maintenance, improve extendability, interoperability and will greatly ease the migration of the FG codebase towards modern best-in-class technologies.
- **DFG** : The FG architecture is a kind of server which maintains a collection KBObjects and their relationships (Subject, Predicate, Statement, Relation, etc...). Each KBObject has the following member variables : created by, modified by, creation date, modified date. The closest equivalent paradigm in Drupal is the Drupal node (DN). A DN represents the smallest unit of information in Drupal which can be edited, version controlled, administrated etc... I propose that each KBObject in FG become a DN. When a user browses to a DN with and HTML browser, all relevant linked DNs can be displayed. For example : Navigating to a Subject DN will display all Statement DNs belonging to it or its inherited Statement DNs,

all ancestor Subject DNs and children Subject DNs, etc... At declaration time, a DN can be customized with extra fields which would be ideal to establish the relevant logical links between KBObjects at DN instantiation or DN modification time. This corresponds directly with the 'foreign key ID' field entry in the SQL table representation of a KBObject as required in FG logic. Thus all the FG source code can be re-used with very little modification. All that has to be changed in JFG code is to include the Drupal-specific database fields in the JPA KBObject-to-Database mapping configuration tags in the Java source code. JFG must properly update the DFG-specific fields and vice-versa DFG has to properly update the JFG specific fields. Concurrency is handled by Drupal with its built-in DN-specific concurrency handler, all that remains is to add that same locking mechanism to JFG to solve the parallel user problem.

(B) Graphical User Interface (GUI)

- **JFG** : The old FG GUI is based on Java Swing components. A decade ago when FG was actively being developed, certain portions of Swing were immature and unreliable, the Editor feature of Swing components simply did not work as advertised and the drag-and-drog simply failed to function. Both of these failings were beyond our control (the fault of the former Sun miocrosystems) and prevented the FG GUI from being editable. We were stuck with importing specially formatted text files to create an entire KB in one step. Fortunately FG was designed with a strict separation of concerns between the Java Swing GUI and the core-logic. I will take advantage of this by eliminating the buggy GUI layer and replacing it with the Netbeans platform. This will eliminate most Swing bugs because the NetBeans GUI components are more mature and powerful. All of the tedious error-prone boiler-plate code for instantiating Swing's TreeView and Table is replaced with much cleaner NetBeans equivalents. In addition NetBeans has new functionality which was weakly emulated in the old FG such as Wizards, Button Bars and Action Button/Menu framework all available with single line instantiations. This will eliminate the need for nearly 80 percent of the FG GUI code! The purpose of this NetBeans migration is very simple : code re-use; Why re-invent the wheel when NetBeans offers a full-fledged application framework. JFG consists of the NetBeans platform in combination with JPA and will allow fully editable KBs.
- **DFG** : The old FG KB web export feature allows any regular HTML browser to view and consisted of static HTML pages and some .class Applet files for creating an instance of the navigation tree which runs on the client-side. The KB was not editable in this environment. The relevant FG exporter logic consists of highly coupled Java code with hard-coded HTML generation mixed up with FG logic code; nearly impossible to modify due to the spaghetti code. The plan is to migrate all this functionality to Drupal which has an amazing assortment of HTML templates, Cascading Style sheets, layout engines and hundreds of Drupal modules for taxonomy and data processing. Shortly after the exporter is converted, the logic for creating and maintaining a modifiable KB will be added, initially as an EDIT THIS PAGE link that users are familiar with from the Wiki world; but eventually the individual subjects and statements on the currently view page or KBObject DN will be individually click-able and editable.

Drupal 6 and 7 have lots of taxonomy modules. Drupal 7.0 was released 6 months ago and features native support for RDFa. Empirical evidence shows that RDFa is the fastest growing markup on the web now at 3.6% and growing. I will make FG interoperable with RDFa. In all probability we may leverage multiple languages for the distinct parts of FactGuru for which each language has it strengths :

1. **Java** : Ubiquitous server side language, enterprise mature, platform independent. Awesome NetBeans platform (why re-invent the wheel).

2. **Drupal (PHP, JavaScript)** : Ubiquitous, powerful and flexible web GUI, again why re-invent the wheel?
3. **CouchDB (Erlang)** : Support for massively parallel and distributed knowledge bases. Strong Prolog pedigree which will aid with incorporation of future logic plug-ins such as conceptual graphs, inferencing engines etc... Erlang also has native support for inter-operability with other languages such as Java. Major vendors are already making use of Erlang.

I will be making the JFG UI appear similar to the current Web UI because the LAKE lab had already agreed that this format is the easiest to read back in 2001. For the sake of desktop users not interested in client-server, JFG will have no dependence on DFG however the converse will not be true. For simplicity and ease of programming, DFG will invoke JFG methods and access JFG datatypes. (Unless we decide to create a pure DFG LAMP version for the Drupal purists unwilling to run server-side Java, but that is a lot more work and has maintenance problems). Naturally JFG is model-centric and DFG is UI centric. JFG will have a basic HTML web export format, similar to the current web FG. DFG however, will have a highly flexible set of HTML output options and formats due to the enormity and maturity of the Drupal community. Using this architecture DFG would take on the subordinate role of a JFG plug-in for client-server purposes.

As per Sowa's suggestion I will make it as modular as possible so that people only use or download what they like

1. The user interface for JFG runs inside NetBeans and will comply with the OSGI plugin architecture for standardized enterprise strength modularity for model-centric plug-ins.
2. DFG modularity will leverage the standardized Drupal module API for extensions and plugins. You could extend the system via Drupal modules for web UI-centric enhancements.

I have been studying Erlang and CouchDB as suggested by John Sowa. I obtained all the necessary educational materials etc... Erlang and CouchDB are well suited to FG. CouchDB is interesting because of replication : its like distributed version control systems, there is no need for a central server, all distributed copies of the database keep themselves consistent transparently. You could put into the database : CG, CT, or FG Facts, Subjects, Verbs, NL, HTML with the added advantage that there is no fixed schema like in RDBMS. RDBMS have schema maintenance nightmares resulting from either software requirements escalation, switching to a new version of the database engine or switching to entirely new databases. CouchDB can be augmented with various plugins such as the Java Lucene search engine (Erlang plays nice with other languages). CouchDB is more similar to an OODBMS rather than an RDBMS in its design.

The pi-calculus approach for software development is based on parallel processes that communicate via message channels. The messages themselves consist of channels which allow a dynamic graph topology between processes. Erlang has the same philosophy, its naturally parallel as opposed to shared memory based systems which require problematic locks and were developed before the internet, and based on unsuitable sequential computation models. Erlang was designed to be the parallel distributed language 'of' the Internet rather than other languages such as Java and PHP which are designed 'for' the Internet. I found process caluculi such a fascinating concurrency model that I created a small FG knowledge base with 320 facts in text input format (the KB includes some semi-related context information such as the difference between sequential programming models and concurrency models.)

Prolog IKS is VivoMind's implementation of pi-calculus. We need to get on the ball regarding Prolog IKS, for example is it available to us?

Another interesting integration option :

Our system should interoperate with current standards such as conceptual graphs (CG), fortunately, a group of CG researchers in Europe created COGUI a Java NetBeans module which provides us with a highly convenient integration route. I am planing to make JFG just another simple NetBeans module which will interoperates with COGUI. You could easily drag JFG content and drop it into COGUI and vice-versa. Both GOGUI and JFG will be in the same NetBeans UI, simplifiying deployment and making a much more powerful system. When Aaron saw FG a decade ago he said it could be a great GUI for Sowa's CG.

Here is recent book written by the authors of COGUI :

Springer - Graph-based knowledge representation - Computational foundations of conceptual graphs (M.Chein, Marie-Laure Mugnier 2008)444p 1848002858 Q387.C34_2008

the book has website: http://www.lirmm.fr/gbkrbook/

- COGUI is the Java GUI for CG
- GOGUI is open source
- COGUI has excellent documentation
- COGUI has XML export format cogxml
- COGUI is interoperable with RDFs
- COGUI is created inside NetBeans using simple NetBeans modules

FG+CG is very exciting!

Also what will be the role of Link Grammar, you said you would be using it?