IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks

Version 1.1

Getting Started Guide

Contents

1 Introduction

IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is a tool for building Web 2.0 applications. Using the miner is simple, and in many cases useful applications can be created as easily as a quick hack. However, the algorithm scales up to enterprise solutions just as easily. All you need to do is pass the right data describing the problem to it. At the same time, the IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is a tool for developing innovative intelligent methods for Web 2.0.

Web 2.0 is about people, the things they create and do. People form outcome-focused teams, ad hoc knowledge/value-focused communities of practice are emerging, reconstituted and dissolved dynamically with the run of time. People create digital artefacts such as emails, documents, ontologies, and blogs. These digital artefacts are interconnected explicitly, by hyperlinks, and indirectly through concepts mentioned in free texts, and also through their creators, those who read/view them and the activities performed by these people using these resources. People are involved in activities such as organising a conference or a trip, which utilise many aspects of these diverse virtual networks. So software or services must be easily adapted to the needs of the individual and also to the task at hand. The miner is flexible enough to allow the applications it powers to be adaptive to these needs and to deliver personalisation to the user.

Most of the software for Web 2.0 needs must be able to perform elements of social computing, semantic processing, and activity centric computing. IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is a library which, through a simple unified API allows users to create solutions for social computing, semantic text processing, and activity centric computing. This simplicity is achieved by exploiting a highly customisable method of activation spread on networks of nodes, which in our implementation can be used for soft clustering and fuzzy inference on multidimensional networks of people, the things they create (digital artefacts such as documents and concepts) and the things they do. This can even be made sensitive to temporal events for workflow management by treating time as a linear chain of timestamps linked to people, events, and artefacts, within the greater network.

In this Getting Started document we'll describe:

- How to install and run IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks
- The basics of how to use the API
- How to use the miner to extract the most important concepts/ideas/keywords from a text

IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks provides semantic processing both for ontology- and instance-based semantics. For example, one can use ontologies such as MeSH to power the lexico-semantic analysis performed by the miner. In this way IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks can be used to perform ontology-based semantic text processing (including term disambiguation and semantic tagging). This might be used for automatic generation of meta tags for use in semantic web applications or in semantic desktop applications (e.g. social semantic desktop which is under development in the EU project NEPOMUK[1]).

In this Getting Started Document we'll also outline what will be described in more detail in future releases and links to research projects which use this technology:

- How to use IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks for social applications (such as community based tag recommendation in collaborative tagging systems)
- How to use IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks for activity centric computing

[1] http://nepomuk.semanticdesktop.org

2 System Requirements

The distribution package contains the API and a lightweight GUI application (galaxy.exe) built on Eclipse which allows you to see some of the functionality in action, and to build and test your own semantic resources. Depending on your needs you may require one, or both of these components. The system requirements differ slightly for the two as described below. The UIMA and ICU4J requirements only apply to use of the API. The required dependencies are bundled with the GUI application.

2.1 Operating System

API

Use of the API is operating system independent. IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is written as a set of Java libraries which you can use from for own applications. Details of how to do this can be found in Section 6 and in the Javadocs supplied.

GUI

The GUI application supplied is compiled only for use on Windows platforms. Compatibility has been tested on Windows 2000, XP and Vista.

Other Operating Systems

As mentioned above, use of the API is OS independent. The GUI application is built on the Eclipse framework, so it can be ported to any OS supported by Eclipse. In the current release we only generate an executable for Windows, but in future releases we plan to provide this application for other platforms. If you require the GUI compiled for any other platform, please contact us with details of the platform and we will compile an executable to suit your needs.

2.2 Java

The IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks libraries are written in Java. The required runtime is Java 1.5.

2.3 UIMA

Using the API requires Apache UIMA version 2.2.2 or later.

2.4 ICU4J

Using the API requires ICU4J version 3.6 or later.

3 Installation

To install the Galaxy GUI application:

1. Download the `galaxy.zip` file
2. Unpack the zip file to a suitable directory
3. Launch the application (`galaxy/galaxy.exe`)

To use the Galaxy API:

1. Download the `galaxy-api.zip` file
2. Unpack the zip file to a suitable directory
3. Galaxy API is provided as a set of Eclipse plugins/OSGi bundles. To use them in Eclipse IDE, add `galaxy-api` folder to the target platform. Alternatively, plugin jar files from `galaxy-api/plugins` folder can be used without Eclipse/OSGi by adding jar files to the classpath of an application.

4 Tutorial

In this section we provide a brief tutorial on using IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks for the first time. The aim is to provide some familiarity with its capabilities through some basic examples. This is, of course, only the tip of the iceberg. IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is in no way limited to these types of applications, as mentioned in the introduction and will be further discussed in Section 5, once the problem can be described using a graph, the miner can provide an efficient and scalable solution to suit your particular needs.

4.1 The GUI Application

This distribution contains an Eclipse application which provides a graphical interface to the basic functionalities of the libraries. Using this application you can build data into lexico-semantic dictionaries, explore the underlying multidimensional network, perform analysis on text, and perform some basic investigation into what additional information can be inferred from the underlying network.

Compiling and Working with the Sample Data

This distribution comes supplied with a small dataset and some sample texts. The dataset describes information about a fictional company including management structures, products, locations, company terminology, etc. The sample texts are emails relating to the company's business. To work with this data it must first be compiled into a lexico-semantic dictionary (*.lex.dic file) and semantic network (*.sem.dic file) for the miner to use. This process is described in Section 7.4, as well as how to load the compiled resources into the viewer. The dataset supplied in the distribution is stored in a file called `demo.xml` in the `data` directory of the distribution. The corresponding configuration file for compiling and loading the data is `demo.config.xml` located in the same directory.

Exploring the Data Once the resources are loaded the left pane of the viewer is populated with data. This side of the view allows you to explore the semantic network. The "Semantic Trees" tab displays any data in the network which conforms to a tree structure. Switching to the "Types" tab displays all the data (including anything which does not conform to a hierarchical or tree-like structure) in a flattened form. Both of these tabs allow you to filter what you see based on the node type. Clicking on any of the nodes in this pane will populate the pane on the right with data about that specific node (see Figure 1). Clicking on a node name in the

"Links" section of this right hand pane will navigate to the information for that node and will also update the view in the left pane.

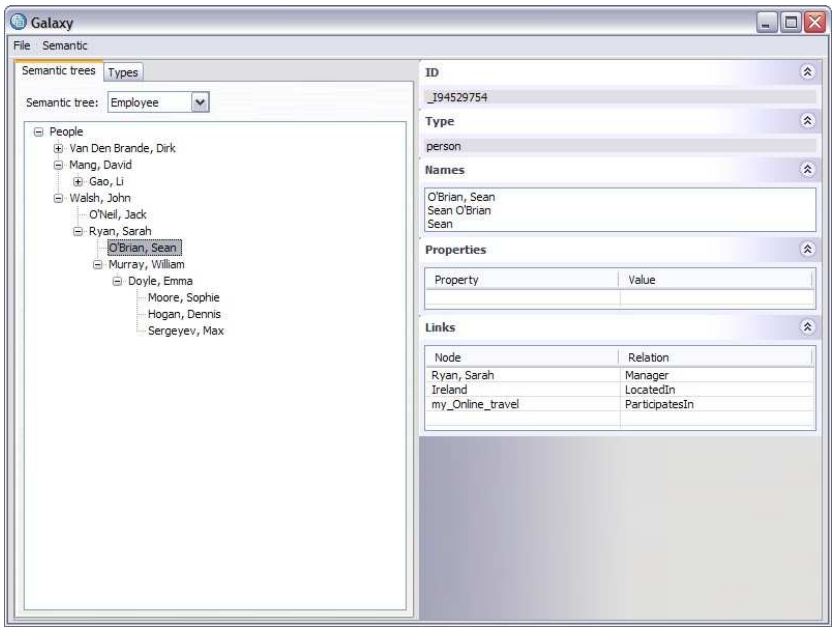

Figure 1: Exploring compiled data

Analysing Text With a lexico-semantic resource loaded you can now start to analyse text. To do this open a text file in the GUI.

- From the "Semantic" menu choose "Open text file"
- A file selection dialog opens, here select the text file you want to analyse, for the purposes of this example select one of the `email.txt` files contained in the data directory

The text will open in the upper left portion of the view. The miner will immediately process the text and display a ranked list of foci opposite the text pane.

In the text pane you will notice that certain terms in the text have been highlighted. These are lexical expressions corresponding to concepts in the underlying network of concepts. In the demo files given, you will notice that the concepts detected include, names, places, products, and even concepts like happiness or urgency which are expressed in the text. These term mentions are then aggregated with other relevant concepts (which are not necessarily mentioned in the text) and ranked by a type of spreading activation method (described in Section 5). This is the ranked list which appears on the right. These are the foci, or concepts which

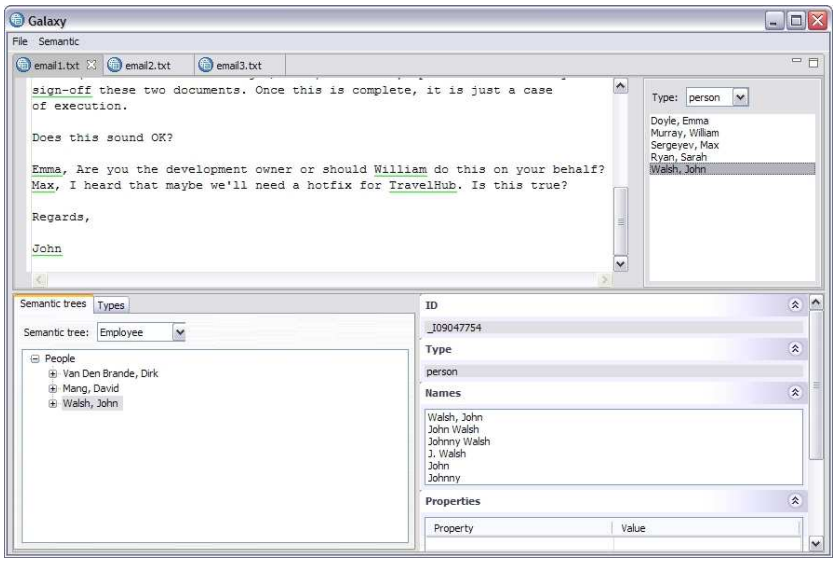

Figure 2: Analysing text

are most relevant to the text currently displayed according to our algorithm. This list can be filtered by node type, to for example, find the most relevant person (in the underlying socio-semantic network) to talk to about a particular document.

For example, notice that if you open `email1.txt` with the data from `demo.xml` loaded that even though the mail is from "John Walsh", he is the *least* relevant person to the topic of the discussion that the miner finds (see Figure 2).

Disambiguation of ambiguous terms (where different nodes are associated with the same lexical expression) is done automatically by the miner, and if you are typing text in the editor, focus finding and disambiguations are all performed on the fly. For example, if you analyse the remaining email text files in the data directory (`email2.txt` and `email3.txt`) you will notice that the lexical expression "Dennis" is ambiguous in the underlying network. This lexical expression can refer to either node 951908672 corresponding to "Dennis Watson" or to node I94446754 corresponding to "Dennis Hogan." However, the miner has correctly disambiguated the term in two different contexts. In `email2.txt` it maps to "Dennis Watson" and in `email3.txt` it maps to "Dennis Hogan." The algorithm performs this disambiguation at run time using a spread of activation technique (described in Section 5) taking into account the other nodes which are "activated" by the other lexical expressions found in the text.

Inference At this stage you will have noticed that the text pane in the GUI allows you to do more than analyse text files on disk. You can enter and edit text in

this pane in the same way you would in a normal text editor and it will be processed on the fly with foci and ambiguities processed as you type. You may also have noticed that if you explicitly type the name of a node (or the lexical expression corresponding to it) in the text editor that it is "activated" and usually appears in the right pane as a one of the ranked foci. By combining these two facts and using a blank file the GUI can be used to draw some very basic inferences from the data in the underlying network. For example, returning to the `demo.xml` data again, if you type in some names and a topic of interest the ranked list of foci can then be filtered to reveal who has the most expertise on this topic. Entering the following text "Emma Max Jack DOU" and filtering by nodes of type "person" reveals that "Emma" is the expert on this topic in the group (Figure 3).

Figure 3: Basic Inference

Even though there is no direct link in the data associating the nodes named "Emma" and "DOU", IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks has been able to determine a degree of closeness between the two based on the cumulative effect of activating the other nodes. This process can be used to suggest additional links to the user, to recommend related artefacts, and even to suggest a course of action. This simple example is purely illustrative, the real expressive power of this capability becomes apparent when it is used in conjunction with larger more descriptive data. For example, we have used this capability to navigate and annotate semantically-enabled networks of people and associated objects [Kinsella et al., 2007]. The limits of the capabilities of this inferencing are dependant upon the type and structure of data in the underlying network. By adding more information and more dimensions to describe the scenario you empower the LanguageWare Miner for Multidimensional Socio-Semantic Networks to infer more about the "world" which your data describes, and so enable it to increase the capabilities, consumability and innovation of your applications which are built on this technology.

5 How it Works

5.1 What is IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks

IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is a library of tools which can be used to process information represented as multidimensional networks, such as social and semantic networks, to extract data about known nodes and infer new information, links and relationships based on the existing knowledge encoded in the network. The miner also provides lexico-semantic mapping from term mentions in text to concepts in a network of concepts. It provides a wide range of functionality necessary to enable the user to build socio-semantic applications, in a single highly customisable technology. The performance of the miner is scales up well with performance in the region of less than 200 milliseconds on networks of hundreds of thousands of nodes on a typical desktop PC.

The combination of soft clustering and fuzzy inference used in the miner is lightweight, scalable and adaptable to traditional socio-semantic tasks such as expertise location as well as tasks from other areas like workflow management.

The scalability and customisable nature of IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks enables you to increase the consumability of socio-semantic and web 2.0 applications through its speed of performance and adaptive nature. Its scalable nature then means it can provide an enterprise level solution just as easily as a quick desktop hack.

5.2 Ontologies

IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is, as the name suggests, a multidimensional network miner, however, because it is currently mostly used for ontology-based analysis of text (eg. the medical ontology MeSH) for a range of different activities, we will discuss ontologies here briefly in the context of multidimensional networks for socio-semantic applications.

For the purposes of socio-semantic applications, an ontology is a collection of nodes which are networked together in a meaningful way. In this way, nodes represent concepts or semantic meanings of a particular “thing” and links connecting nodes represent relationships between nodes. There are no constraints on what concepts and relationships can be represented, for example company management structures, product components, semantic relations between words such as synonymy/hyponymy/meronymy, and any other situation where there is a relationship between two concepts or things can be represented using linked nodes. Figure 4 shows a very simple ontology pertaining to the automotive industry.

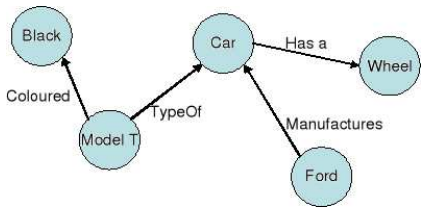

Figure 4: Simple Automotive ontology

Representing data in this way, as a network of interconnected nodes, lends itself to many types of data for many different applications. In a social context they can be used to analyse peoples relationships and interests or to find domain experts. Temporal events can be modelled using a timeline of events embedded within the greater network. Even things like shades of colours can be modelled and accounted for in this way. Likewise, from analysing such networks we can infer new relationships based on existing linkage which can be used (for among other things) to recommend interesting documents, web pages and people based on a persons existing connections.

Often, there are relationships outside the scope of a particular graph which apply to the nodes it contains, and which link to nodes in a different graph. For example, a company management structure graph shows nodes which represent people and their reporting relationships. However, these people have social connections to one another, and so too to people outside the scope of the original graph, for example in another company or organisation (Figure 5).

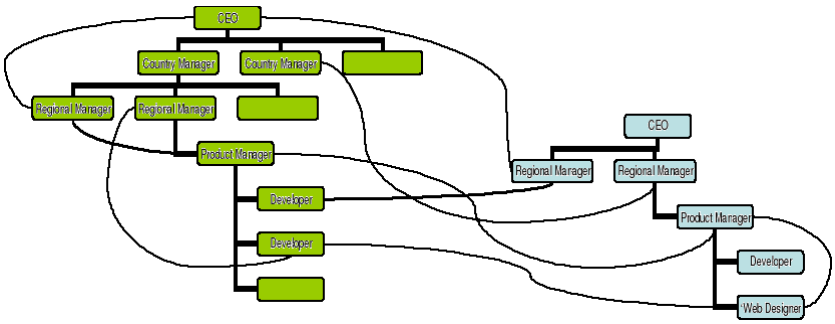

Figure 5: Interconnected Ontologies — Multi-Dimensional Network

Taking this type of connectivity into consideration requires data integration in a composite network which accounts for the different dimensions of the data and also the task at hand. Multidimensional networks of this nature allow applications to draw on much wider context and so to address a wider range of problems with

greater ingenuity than those which only consider a single aspect or dimension of the socio-semantic space. Multidimensional networks, such as these can be used to represent instance-based semantics like collections of documents manually annotated with tags, keywords, or arranged in a system of sub-folders. This aspect combined with the expressive nature of ontologies means that applications which can exploit these networks are in a strong position to deliver innovative new solutions to problems in the socio-semantics and workflow management.

5.3 Mining Multi-Dimensional Networks

Multidimensional networks are data aggregators which combine data from a multitude of data sources. The nature of multidimensional networks means that they include diverse data types, they are often very large, and can change with time. This can make it difficult to find useful information in or do anything useful with such a large dynamic data source, unless you have an efficient scalable algorithm to process the data contained in such a multidimensional network.

Traditional graph analysis techniques present some possible solutions like cluster analysis which groups nodes together as a unit based on some common trait or distance measure. But clustering nodes in this way is vague, in that it reduces nodes to collections, thereby losing finer details and distinctions which often contain useful information. It is also inefficient at processing multi-dimensional networks, as described above, because depending on the task different analysis metrics are necessary and also changes to the data mean that the whole network needs to be repartitioned.

There are a wealth of efficient algorithms for traversing graphs and networks of nodes given a starting point. However, from the point of view of socio-semantic applications and workflow management this is an unsuitable approach as the focus is too narrow (node to node connections instead of the “bigger picture”) and it is also quite process intensive at run time.

Our solution to this problem is similar to the method of spreading activation used in associative networks, neural networks and, more recently, in semantic networks. This method is more abstract than a network traversal, nodes could be said to be grouped together but unlike traditional clustering methods described above, IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks performs a “softer” clustering which retains the finer distinctions between nodes.

For any given request or query, IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks is only concerned with a small subset of the nodes in a multidimensional network, those which are passed to it as starting points. These initial nodes (which might be widely spread across the network) are given an activation level according to their importance to the query (eg. important concepts from a document), then this activation propagates outward from these

nodes along all links. Taking an analogy of light emanating from a point, It works by first illuminating the nodes relevant to a query, then this light spreads through the network illuminating nearby nodes, but to a lesser extent as it travels. The light (activation) can accumulate in nodes which are "illuminated" by multiple nodes and once the process is complete the node(s) with greatest activation level is deemed to be the focus or most relevant concept to the query. This gives us an analysis similar to "fuzzy clustering", dealing with a (changing) sub-network based around a set of nodes within the network provided to the miner, and finds a focus (or foci) relative to those nodes and dependent upon the network topology and the user's constraints on how propagation around the graph can happen. The focus found by the miner is similar to finding a central node or concept for the given sub-graph. However, dependent on the starting nodes, the users conditions and graph topology multiple foci or no focus may be returned. In this way it does not return a central concept or focus unless one can be found which is close to the starting nodes.

All factors regarding the spread of activation through the network, decay rate, link strength, link direction, etc., can be controlled through a series of parameters set in a configuration file, this is discussed in Section 7.3. Tuning these parameters means IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks can be adapted to give different results depending on the application or the users preferences.

The spread of activation acts locally, centered around the nodes which receive initial activation. This is part of what makes the method so efficient, by following the principal of locality in software development. However, if the activation propagates far enough through the network, by means of particularly strong activation, low decay rate or otherwise, it is possible to model not only micro- but also mezzo- and macro-level phenomena.

This method of analysing multidimensional networks is lightweight, scalable and adaptable to traditional socio-semantic tasks such as expertise location as well as tasks from other emerging areas like workflow management. In short, once you can describe the problem and constraints using a graph, IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks can help solve the problem. And it scales up to provide an enterprise level solution just as easily as a quick desktop hack.

6 The API

Full Javadoc is available in the distribution package; we describe the main classes and their usage here.

6.1 Lexico-Semantic Classes

This section describes the main classes of the lexico-semantic module. These classes are available in the `com.ibm.dltj.ls_1.5.0.200711301204.jar` file[2]. The DLTSemanticDictionary<T> template is the entry point to a lexico-semantic dictionary. A lexico-semantic dictionary consists of the following template classes:

Node set (NodeStock<T>)

Holds all nodes and provides a reference to each node (NodeReference<T>). This template class has two implementations:

- NodeBag<T> stores nodes in an array which grows in size when adding more nodes. This stock allows the duplication of same node instances.
- NodeSet<T> stores nodes in a hash set. This stock excludes the duplication of same node instances.

Set of views (NodeView<T>)

Provides some information on nodes in the stock. There may be zero or multiple views in a dictionary. There are three types of views:

LexiconView<T> this view links lexical entries to nodes from the stock. Lexical entries are not unique. i.e. the same lexical entry can be attached to multiple nodes and the same node can have more than one lexical entry. There are two implementations for lexicon views:

- A character trie implementation can be used for longest-common-prefix lookup.
- A hash implementation uses a hash map to store lexical entries. It can be used for exact-match lookup.

SemanticView<T> This kind of views links nodes with each other via a semantic relation. There are four types of semantic views:

[2]The version number (after the underscore) of this file name will change in future releases but the overall name will remain the same

- SemanticTreeView<T>: a tree hierarchy
- SemanticGraphView<T>: an undirected graph
- SemanticDigraphView<T>: a directed graph
- SemanticDigraphViewSet<T>: groups a set of views from the above three types and provides a (read-only) access to them as if they were one directed graph.

IDView<T> associates a unique ID with each node. There are two types of ID views:

- IDStringView<T>: where an ID is a String. This retrieves a node in the form of a NodeReference<T> given its ID.
- IDStringBidiView<T>: same as the IDStringView<T> but adds a bi-directional view of IDs (i.e. given a node reference, we can retrieve the ID of this node).

6.2 Node Types

The generic type T represents the type of nodes. It can be any kind of object. It is however recommended to make nodes subclasses of the GraphNode. As a default node type, we provide DLTGraphNode which is a subclass of the GraphNode and provides the following properties:

Name The name of the node

Label A label for the node; that label can be used in UIs to display information about the node.

Type A String indicating the type of the node.

Properties Properties in the form of (key,value) pairs.

Users are free to use the subclass DLTGraphNode to provide whatever information they need in the node. The Semantic API offers the NodeReference<T> template to create pointer references to nodes:

6.3 Creating and Managing Lexico-Semantic Dictionaries

This section explains how to create lexico-semantic dictionaries using the API.

Instantiating a new dictionary

A dictionary is created via a `createDictionary()` factory method by specifying the URL of the resource (dictionary).

```
URL url = new File("dictionary-ls.dic").toURL();
SemanticDictionary<DLTGraphNode> dictionary =
DLTSemanticDictionary.<DLTGraphNode>createDictionary(url);
```

Creating a dictionary stock

Once your dictionary is instantiated, a stock should be first created to hold nodes.

```
dictionary.createsStock(feature);
```

The feature parameter determines the implementation details of the created node stock. It can take two values:

DLTResourceFactory.STOCK_BAG Bag implementation. i.e. an array which grows in size when adding more nodes

DLTResourceFactory.STOCK_SET Hashed implementation.

Creating views

You create a view by using the `SemanticDictionary.createView()` method.

```
dictionary.createView(id, feature);
```

It requires two parameters:

- id: a unique ID for the created view
- feature: determines the type of the view

The parameter **feature** takes the following values:

Feature Value	**Created Lexicon**
Lexicon Views	
DLTResourceFactory.LEXICON_TRIE	Character trie implementation
DLTResourceFactory.LEXICON_HASH	Hashed implementation
Semantic Views	
DLTResourceFactory.SEMANTIC_TREE	a tree hierarchy
DLTResourceFactory.SEMANTIC_GRAPH	an undirected graph
DLTResourceFactory.SEMANTIC_DIGRAPH	a directed graph
DLTResourceFactory.SEMANTIC_SET	a grouped set of one or more views from above views

Table 1: Feature Values

Semantic views have a `reverse()` method that returns a view which is the reverse of the original view (i.e. direction of edges is reversed).

You can build an ID view using `SemanticDictionary.buildIDView()` function.

```
IDView<DLTGraphNode> idView = dictionary.buildIDView();
```

There are methods available for each type of view. The Javadoc describes each method in detail. For samples, refer to the sample files provided with LanguageWare or download them from the IBM site at http://languageware.mul.ie.ibm.com/downloads/jFrost/LW61/samples/.

Saving and loading the dictionary

Dictionaries can be saved to or loaded from a dictionary file by using the load and save methods of the dictionary class. This will save or load a dictionary to or from the URL specified when creating the dictionary.

```
dictionary.load();
dictionary.save();
```

6.4 Running a Semantic Processor

The semantic processor is the object that performs the semantic analysis of your documents. The following sections explain how to configure and run a semantic processor.

Initializing the semantic processor

A semantic processor must be initialized with contextual information: it must be given the name of the dictionary to use, the class of the focus determiner to use, lists of words to ignore in the analysis, etc. This is done through the SemanticProcessorContext class.

```
SemanticProcessorContext context = new SemanticProcessorContext();
context.dictionary = new File("dictionary-ls.dic").toURL();
context.focusDeterminerClassName =
    "com.ibm.dltj.ls.mining.impl.FocusDeterminerNet_Luminous";
context.focusDeterminerConfig = new File("config.xml").toURL();
context.genBlacklist = new File("generic-black-list.txt").toURL();
context.specBlacklist = new File("specific-black-list.txt").toURL();
```

The SemanticProcessorContext class has the following properties:

Property	Description
dictionary	URL pointing to the location of a lexico-semantic dictionary
focusDeterminerClassName	Fully qualified Java class name of a SemanticProcessor implementation
focusDeterminerConfig	URL of the XML configuration of semantic processing, described before
genBlacklist	URL pointing to a text file containing newline-separated generic words exclusion list
specBlacklist	URL pointing to a text file containing newline-separated specific words exclusion list

Table 2: Properties of the SemanticProcessorContext class

Instantiating and executing the semantic processor

A semantic processor must be created with the above context and it must be opened to load the needed resources.

```
SemanticProcessor processor = new SemanticProcessorImpl(context);
processor.open(context);
```

The semantic processor is now ready for use and can analyse pieces of documents (text chunks). The text is input in the form of an ArrayList of Strings. This means that preprocessing may be needed to transform a document or document parts into this array.

You might use the LanguageWare lexical analysis to build this array since the text chunks are typically tokens or multi-word units - these are lexical expressions that have been identified in text and can potentially be mapped onto a concept.

```
ArrayList<String> textchunks = new ArrayList<String>();
// populate the textchunks with tokens or MWUs from Lexical Analysis
...
SemanticProcessorResult result = processor.getFoci(textchunks);
```

Working with the results

The semantic processor returns an object of the class SemanticProcessResult. This class has two properties:

fociList a list of ResultNodes describing the focus of this piece of text

disambiguationList a list ResultNodes corresponding to the input list. For each input String we have either null (if the word could not be disambiguated) or a ResultNode describing the disambiguation of this String)

Each ResultNode contains the following parameters:

ID the id of the node described by this ResultNode

Label the label of the node if it exists

Score the confidence in this result.

Closing the semantic processor

Once the semantic analysis is done, the processor should be closed.

```
processor.close();
```

6.5 Known Limitations

The current implementation of the lexico-semantic dictionary is not thread-safe. Developers will need to implement their own locks to avoid modifying resources while they are used for semantic analysis.

7 Creating Semantic Resources

The lexico-semantic dictionaries used by IBM LanguageWare Miner for Multidimensional Socio-Semantic Networks have a unique format and content. This type of dictionary is used to identify concepts in texts and to disambiguate them with respect to the context (the other concepts mentioned in the same text). As a result, this data is built into two different objects:

Lexical dictionary (*.lex.dic) This lexical dictionary enables concepts to be spotted in texts. It contains the various surface forms for a particular concept. For example, IBM is a concept that has the following lexical representations: IBM, International Business Machines, I.B.M, IBM Corporation. This lexical dictionary is compiled with a multi-word unit (MWU) format since the lexical variants of a term can contain multiple words that may inflect.

Semantic network (*.sem.dic) This binary object stores the concepts and their relationships as nodes and links. It is then traversed during the analysis. When the lexical analyser finds such a concept (one of the referenced surface forms), it uses the lemma gloss to then find entry points into this semantic network. The system can then explore the concepts that are related, so as to provide some level of semantic analysis and disambiguation.

7.1 Lexico-Semantic Dictionaries

Lexico-semantic dictionaries are used to provide semantic disambiguation. Semantic disambiguation is the process of finding the right concept given the context. For example, the word “Gates” is very ambiguous; it might refer to the gates at the entries of buildings, it might refer to the Microsoft founder “Bill Gates”, it might also be the name of a computer game. To be able to decide, it is possible to create an ontology that describes concepts and how they are linked together.

For example, it is possible to describe the concept of a company and list Microsoft Corporation as an item in this category. And it is possible as well to define the concept of Executive Manager or CEO and to list Bill Gates as a member of this category. It is also possible to associate variants or synonyms to these members. For example, Gates would be a logical synonym for Bill Gates and Microsoft would as well be an acceptable synonym for Microsoft Corporation. Then of course, we can define relationships between our concepts. Typically a CEO “leads” a company and of course we can associate Bill Gates and Microsoft Corporation using this relationship. We can as well set a weight to this relationship. Typically, the relation between a company and its Executive Manager is a strong one. The link between a company and a country by an “operates-in” relation would for example be weaker now that companies are very global.

What is the purpose of describing these concepts and relationships? The purpose is to use these links, follow them and, when some concepts co-occur in a text, confidently associate the right concepts behind words. Typically if both Microsoft and Gates co-occur in a document, it is very likely that Gates is definitively referring to Bill Gates, the Microsoft CEO. However, if the United States and Orange co-occur in a text, we have difficulties associating a company tag to Orange, because that the link between a company name and a country is weak. And typically that is the best approach because a text could very well be about the production of oranges in the United States.

7.2 XML format for lexico-semantic dictionaries

Semantic graph

All of the information in the lexico-semantic dictionary is stored in semantic graphs. A semantic graph consists of nodes and links among these nodes. A node could describe a thing, a concept, an item, etc. The link between nodes specifies a relation.

Nodes Each node must have a unique *id* attribute which takes the format of a case-sensitive string, without having to be human readable. A node may have the attribute type, as a case-sensitive string. It may also have a label. The label is typically the information displayed in user interfaces when we refer to this object. Example:

```
<node id="John Smith" type="person">
    <label text="John Smith"/>
</node>
```

Properties Nodes may also have additional information, given a list of properties. Each property is composed of a name attribute and a value attribute; both being strings. The name attribute contains the name of the property, and the value attribute holds its value. Properties can be used for anything that is not a link to other nodes. For example:

```
<node id="John Smith" type="person">
    <property name="address" value="Somewhere"/>
    <property name="phone" value="1234567"/>
</node>
```

Relations A semantic graph is made of links which connect the nodes. A graph link describes a relation between two concepts. All links have a subject, relation name and object. The subject is the node in which this relation is described. The relation name and object are specified with XML attributes. For example:

```
<node id="Mary Allen" type="person">
    <link rel="manager" id="John Smith"/>
</node>
```

Relation types There are three different relation types:

Directed graph the most general relation type.

Non-directed graph used for symmetric relations.

Tree for hierarchical structures, taxonomies, etc.

The following sections explain the three types of relations in detail.

Graphs and Trees

Directed Graph A directed graph is the most general relation type; it has a name (the relation name) and there is also a name for the inverse relation: For example:

```
<digraph name="manager" reverse="manages"/>
```

Logically, directed graphs imply that if there is a link between A and B through the relation R, there is a link between B and A with the reversed relation of R. In our sample, if Marys manager is John, then John manages Mary. With a directed graph, the following 2 examples are equivalent:

```
<node id="John Smith">
    <link rel="manages" id="Mary Allen"/>
</node>
<node id="Mary Allen"/>

<node id="John Smith"/>
<node id="Mary Allen">
    <link rel="manager" id="John Smith"/>
</node>
```

Non-directed graph Non-directed graphs are specified with the XML tag graph and the attribute name. For example:

```
<graph name="worksWith"/>
```

Non-directed graphs give symmetric relations; i.e., the link from A to B implies the link from B to A. Links are created just like for digraphs, using the link tag. In our sample, if John works with Alan then that means that Alan also works with John.

Trees A tree is specified with forward and reverse relation names and an id of the root node:

```
<tree name="manages" reverse="manager" root="IBM"/>
```

The forward relation is used for navigating from the root to the leaves, while the reverse relation is used to navigate from the leaves to the root. Links can be added as for digraphs with the link tag, or can be specified using nested XML nodes as in the example below. Notice that the link name is just given once for all items, in the top node. This aims at avoiding some verbosity.

```
<node id="IBM" type="company" rel="manages">
    <node id="John Smith" type="person">
        <node id="Mary Allen"/>
    </node>
</node>
```

In this example, we specify the hierarchical management structure of the “IBM” company. “John Smith” who is a person is managing “Mary Allen” who is also a person. Notice that the node type is inherited by the nested XML nodes. In the example, we do not have to specify that Mary Allen is a person. This is inherited from the previous node “John Smith”, by default.

Example of a Full XML Structure

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<lsxml>
    <relations>
        <tree name="manages" reverse="manager" root="IBM"/>
        <graph name="worksWith"/>
```

```
    </relations>
    <nodes>
        <node id="IBM" type="company" rel="manages">
            <label text="IBM"/>
            <name text="IBM"/>
            <name text="International Business Machines"/>
            <node id="John Smith" type="person">
                <name text="John Smith"/>
                <name text="Smith, John"/>
                <name text="John"/>
                <name text="Smith"/>
                <property name="address" value="Somewhere"/>
                <property name="phone" value="1234567"/>
                    <node id="Mary Allen"/>
                        <name text="Mary Allen"/>
                        <name text="Mary"/>
                        <link rel="worksWith" id="James Johnson"/>
                    </node>
                    <node id="James Johnson"/>
                        <name text="James Johnson"/>
                        <name text="James"/>
                    </node>
            </node>
        </node>
    </nodes>
</lsxml>
```

Note

- A node may have several different lexical expressions.
- Different nodes may be associated with the same lexical expression.
- IDs need to be unique.

7.3 The XML configuration file

A configuration file is needed to use the lexico-semantic resources described above. This config file specifies the linguistic resources to be used, the type of annotations we want (e.g. UIMA annotations), and the Java classes we want to use when customization of behaviour is possible. It also gives specific parameters to some relationships through the strength and changing properties. By setting these properties we can control the behaviour of the semantic navigation and give more weight to some relations, to allow some concepts to be more easily reached than others.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
 <dictionary
    source="example.xml"
    text_processor="jFrost"
    lex_file="example.jfrost.dic"
```

```
    gm_file="example.sem.dic"
    lexicon_view="com.ibm.dltj.ls.Lexicon"
    semantic_view="com.ibm.dltj.ls.Semantic"
    parser_class="com.ibm.dltj.ls.dictbuilder.LSXMLParser"
    node_class="com.ibm.dltj.ls.resource.impl.DLTGraphNode"/>
 <focus_determiner
    class="com.ibm.dltj.ls.mining.impl.FocusDeterminerNet_Luminous"
    max_weight_mult="0.3" max_distance="4">
  <relation name="manager" strength="0.7" changing="0.5" />
  <relation name="manages" strength="0.5" changing="0.3" />
   <relation name="worksWith" strength="0.7" changing="0.5" />
  <relation name="*" strength="0.4" changing="0.2" />
 </focus_determiner>
</configuration>
```

Dictionary Attributes

Lexico-semantic dictionaries can have the following attributes specified in the configuration file:

source The location of the source XML dictionary file:

- absolute URL/URI
- relative URI
- filename if the dictionary is located in the same directory as the configuration file.

text_source The text processor being used. The default is “jFrost”.

lex_file The lexical dictionary file (generated from the source XML).

gm_file The semantic dictionary file (generated from source XML)

lexicon_view The name of the main lexicon view in the semantic dictionary. The default is com.ibm.dltj.ls.Lexicon.

semantic_view The name of the main semantic view in the semantic dictionary. The default is com.ibm.dltj.ls.Semantic.

parser_class The name of the parser class. If the XML dictionary format is used, then this parameter must have the value com.ibm.dltj.ls.dictbuilder.LSXMLParser.

node_class The node class. If the XML dictionary format is used, then this parameter must have the value com.ibm.dltj.ls.resource.impl.DLTGraphNode.

blacklist_file A blacklist file is a plain text file with one lexical expression on each line. The blacklist should contain lexical expressions from the dictionary which also have significant usage outside the dictionarys specific domain. For example, “University” must be in blacklist for a geographical ontology because its main usage will not refer to the town called University, but to an educational institution.

Attributes of the focus_determiner element

The configuration file can have the following attributes for the focus_determiner element:

class Focus determiner class. The default is com.ibm.dltj.ls.mining.impl.FocusDeterminerNet_Luminous.

max_weight_mult Maximum weight, which is the threshold for the focus determiner algorithm. It must be within the range [0,...,1]. A zero value would mean no threshold and is not advisable. The reasonable range would be within [0.1,...,0.3].

max_distance Maximum distance, which is the maximum number of steps for spread-activation. If the distance between nodes is larger than the threshold, they are considered completely unrelated.

relation Parameters for individual relations:

- **name** Relation name (* is a wild card and means all relations).
- **strength** During spread-activation, a step through the link of this relation type will decrease the weight by this factor. A value of 0 means no spread. A value of 1 means no decrease. Intermediate values are more reasonable.
- **changing** This is the strength that must be applied when the relation type of the previous step in the path of spread-activation was different from the relation type of the current step.

7.4 Compiling the Semantic Resource

Once you have described your data in the XML format described above compiling it into a lexical dictionary and semantic network can be done through the GUI application (galaxy.exe), or from the command line. For full details of how to compile using the command line application see the Javadoc for the LSDictionaryBuilder class.

To compile using the GUI

1. From the “Semantic” menu select “Build dictionary”
2. A file selection dialog opens; here select the configuration file appropriate for your data and click “Open”
3. A progress indicator will show when the semantic resource has been compiled into dictionaries.

To load the newly created dictionaries:

1. From the “Semantic” menu select “Open dictionary config file”
2. A file selection dialog opens; here select the configuration file appropriate for your data and click “Open”
3. A progress indicator will show when the semantic dictionaries have been loaded

References

[Kinsella et al., 2007] Kinsella, S., Harth, A., Troussov, A., Sogrin, M., Judge, J., Hayes, C., and Breslin, J. G. (2007). Navigating and annotating semantically-enabled networks of people and associated objects. In Friemel, T., editor, *Proceedings of Applications of Social Network Analysis*.