

Melita

Manual Vr 1.0

Alexiei Dingli

Department of Computer Science
Regent Court, 211 Portobello Street,
Sheffield, S1 4DP,
UNITED KINGDOM

melita@dcs.shef.ac.uk

July 17, 2003

Abstract

This manual is intended to provide a comprehensive reference for the Melita application. Melita is an annotation tool that exploits interaction between the user interface and Adaptive Information Extraction (AIE) algorithms, in order to help reduce burdens experienced by the users during the training of an AIE system. This has the positive effect of reducing drastically the amount of training required. This manual contains all the information necessary to guide a typical user through the installation phase. It will then illustrate how to use the tool starting from simple tasks and proceeding gradually towards exploring the more advanced features. The document will also give a brief description of how to create an Ontology for the application and will provide some theoretical information for more advanced users.

Acknowledgements

No duty is more urgent than that of returning thanks.

- James Allen

The current work has been carried on in the framework of the AKT project (Advanced Knowledge Technologies, <http://www.aktors.org>), an Interdisciplinary Research Collaboration (IRC) sponsored by the UK Engineering and Physical Sciences Research Council (grant GR/N15764/01). AKT involves the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. AKT is a multimillion pound six year research project that started in 2000. Its objectives are to develop technologies to cope with the six main challenges of knowledge management: acquisition, modelling, retrieval/extraction, reuse, publication and maintenance. An integral part of Melita is Amilcare (<http://nlp.shef.ac.uk/amilcare/>). Thanks to Dr Fabio Ciravegna for providing Amilcare and for all the help he provided in integrating it with Melita.

Contents

1	Introduction	6
1.1	What is Melita?	6
2	Getting Started	8
2.1	The Melita Distribution	8
2.2	System Requirements	9
2.3	Installation	9
2.3.1	Installing the Client	10
2.3.2	Installing the Server	10
	Note 1 - The DOS prompt	11
	Note 2 - Non Windows users	11
	Note 3 - Some suggestions	11
2.4	Running Melita for the first time	13
3	First steps in Melita	14
3.1	Setting a Scenario, as easy as ABC and D	14
3.1.1	Assigning it a name	15
3.1.2	Browsing for an Ontology	15
3.1.3	Choosing the Corpus	16
3.1.4	Dragging the intervention bar	16
4	The Melita Interface	18
4.0.5	The Ontology panel	18
	The Ontology menu	19
4.0.6	Editing NLP Rules	22
4.0.7	The Document panel	24
4.0.8	The Menus	25
	Note 1	26
	Note 2	26
5	Contact details	28
6	Conclusion	29

7	Index A	30
7.1	The Ontology	30
7.1.1	What is an Ontology?	30
7.1.2	Ontology creation for dummies	30
8	Index B	32
8.1	Introduction to Regular Expressions	32
8.1.1	Literal strings	32
8.1.2	Metacharacters	32
8.1.3	Character classes	33
8.1.4	Predefined character classes	34
8.1.5	Capturing groups	34
8.1.6	Quantifiers	34
8.1.7	Boundary matchers	36
9	Appendix C	37
9.1	Part-of-Speech Tags used in the Hepple Tagger	37
10	Index D	39
10.1	The Gazetteer File structure	39
11	Index E	40
11.1	Theory behind Melita	40

List of Figures

2.1	Melita Splash Screen	10
2.2	Setting the priority of the Melita Server in the Windows Task Manager.	12
3.1	The Melita Scenario Manager.	15
3.2	File selection.	16
3.3	A user can adjust the global the intervention level by moving the two knobs.	17
4.1	The Melita Main User Interface.	19
4.2	A concept in the Ontology panel.	19
4.3	The menu which pops up when the user right clicks on a concept in the Ontology.	20
4.4	The Gazetteer editor.	22
4.5	NLP rules per tag.	23
4.6	The editor.	24
4.7	Example of the different kind of annotations.	25
4.8	Documents rankings in Melita	26

List of Tables

4.1	Examples of regular expressions which can be used to match the time "3:30"	21
8.1	Predefined character classes	34
8.2	Boundary matchers	36

Chapter 1

Introduction

Not all who wander are lost !

- J.R.R. Tolkien

If you are reading this document there are probably a number of reasons for doing so. You might have the Melita tool but don't have any idea from where to start! You could have heard about this tool and would like to know more! One can go on listing reasons for ever, starting from the most plausible and proceeding towards the more bizarre science fiction stuff. I'm not going to list them all (mainly due to time limitation) but even though i can't be sure what you expect from this document, one thing i'm certain of, you do know that this document has something to do with this tool called Melita. So lets start by finding out what is this tool...

1.1 What is Melita?

Melita is an ontology-based demonstrator for text annotation. The goal of Melita is not to produce a further annotation interface, but a demonstrator of how it is possible to actively interact with the IE system in order to meet the requirements of timeliness and tunable intrusiveness ([Ciravegna et al., 2002a]). Timeliness refers to the time lag between the moment in which annotations are inserted by the user and the moment in which they are learnt by the Information Extraction system. Normally this happens sequentially in a batch. The Melita system implements an intelligent scheduling in order to keep timeliness to the minimum or practically non-existent in learning without increasing intrusiveness. The Intrusiveness refer to the suggestions which the IE system gives to the user in order to help reduce the burden of annotating tags. Melita is very similar in spirit to MnM (Vargas-Vera et al. [2002]), Ontomat (Handsuh et al. [2002]) and the Gate annotation tool (Cunningham et al. [2002]).

The Melita system is not a tagging tool but rather a show case of different technologies. First of all it tries to implement intelligent user interface. To do so (Hook [2000]) identified four challenges which are usability, development methods, adaptability and maintainability. All of these four tasks are implemented to a certain degree in Melita. The system was designed to be very usable for both expert users and especially for naive users.

The development methods used are based upon modern object oriented principles. They employ robust Java technologies such as RMI, therefore making the application reliable and easily maintainable for future enhancements. Regarding adaptability, the system is able to cope both with naïve users and expert users. It is up to the user to decide how the system should interact. This can be achieved by simply dragging a knob to select the different levels of pro-activity which the system has (More details below).

Secondly, it implements a methodology with the intent to manage the IE process for the users. It was noticed that several steps in the IE process, which till now are done manually can be easily automated and handled all by the system. The main competencies of Melita can be grouped into four groups the Managing task, the Extraction, the Learning and the Information Tagging Autonomously.

The Managing task of Melita first involves a smart way of interaction with the Information Extraction algorithm in this case Amilcare is being used. Most of the IE systems perform learning in batches because it is quite an expensive process (processing wise) to teach the algorithm after every document. There were basically two solutions for this problem, either increasing the processing power of the machine, which is not feasible for most users or make use of distributed computing. The latter was used, and involved implementing a smart Client/Server approach with the IE system in this case Amilcare. A smart wrapper was built around Amilcare so that the services of Amilcare can be offered via a server. In this way all the processing power for learning is taken from another machine, and therefore the user is not impeded to continue working because of lack of processing power. The system is smart because it is capable of detecting whether or not a connection with the Amilcare server exists. If it does not, a local server is launched and the IE algorithm is run on the same machine as a separate server. This is still faster than using a local copy of the IE engine. The reason is that a local copy uses the resources of the same process as the tagging interface whilst a separate server has resources just for itself allocated by the system. The reason for requiring an efficient way of dealing with the learning Algorithm is that during the tagging process, the Melita system uses constant feedback from the IE algorithm to give as much help to the user as possible. Melita uses knowledge generated during the learning and training process to make suggestions to the user.

Chapter 2

Getting Started

Even the longest journey starts with a single step !

- Old Chinese Proverb

2.1 The Melita Distribution

The Melita distribution is made up of three main files. These are ...

The Melita Client - contains all the functions related to the graphical user interface. It takes care of all the interactions between the user and the application. Apart from that, within the interface it also provides an interface directly with the server therefore giving the user the faculty to manage the server through the main user interface.

The Melita Server - is the part of the application that manages the Adaptive Information Extraction (AIE) algorithm. The reason for this separation is due to the fact that every AIE algorithm requires lots of resources in terms of memory and processing power. It would be unfeasible to have both the AIE and the user interface running in the same process, and this for a number of reasons. The two main reasons being that first of all, Melita can still operate without the AIE (even though the AIE is integral to give that added value to Melita). Secondly, if the AIE and the User interface run in the same process, the AIE would probably take most of the resources of the system from the User interface therefor making the usability of the system unreasonable.

The Tools.jar - is a file normally found in the Java Runtime Environment (JRE). The reason why it is included in this distribution is because one of the subcomponents of the AIE algorithm used makes use of this file. If this subcomponent does not find this file or if another version of this file

is found in the JRE (even if its more recent), the AIE will fail to execute. The file is located within the JAR file of the Melita Server. For users installing the system on a windows based machine, this file will be copied automatically.

Although a brief description of why this sort of architecture was used is given in this section, a more detailed discussion can be found at the end of this document in Section 11.1.

2.2 System Requirements

It is important that before even thinking of installing Melita, one should check the system requirements. These are the minimum requirements necessary to obtain a decent performance. If a system has better specifications than these listed down here, then it will obviously enhance the performance of the application.

Melita Client

- Win 2000 Operating System or better (It can also run on Linux but extensive testing was only performed on Windows based systems)
- 128 Mbs RAM
- 800 Mhz CPU or better
- Java Runtime Environment 1.4 or better

Melita Server

- Win 2000 Operating System or better (It can also run on Linux but extensive testing was only performed on Windows based systems)
- 512 Mbs RAM
- 800 Mhz CPU or better
- Java Runtime Environment 1.4 or better

2.3 Installation

This section covers how to setup both the Client and the Server. They are both very straight forward, the only difficulties might arise while setting the server for non-Windows machines but don't be afraid, the greatest pleasure in life is doing what other people believe you can't ...



Figure 2.1: Melita Splash Screen

2.3.1 Installing the Client

To perform the Client installation, you must know one fundamental thing, how to double click using the mouse. If you know how to do that, then you're almost finished. To install the client ...

1. goto the directory where the Melita.jar file is stored.
2. double click on it.

That's all!! It wasn't difficult!! At this stage, you should see the main Melita splash screen (See Figure 2.1) signifying that the Melita application is working fine.

2.3.2 Installing the Server

In order to setup the server, one must do two things ...

1. open a DOS prompt and go to the directory where the MelitaServer.jar is located.
2. type

```
java -jar MelitaServer.jar
```

To check if the installation went fine, a user can look at the DOS prompt where the Server was run. If the message **Amilcare Server Online !** appears, then it means that everything is working fine.

Note 1 - The DOS prompt

There is no need to open the DOS prompt in reality. A user can just double click on the MelitaServer.jar and the program should run automatically. The only reason why it is suggested that the program is run through the DOS prompt is so that any diagnostic messages which the Melita Server prints out are visible by the user. If it is not run via the DOS prompt, the server will live as a process in the system and the user will have no way of seeing the messages returned by the server.

Note 2 - Non Windows users

Although the code is platform independent, some setup procedures are not. Because of this, i will illustrate the setup steps required if a user would like to install the server on a machine which does not run windows.

1. Find the path of the current Java Runtime Environment. It will be used in later steps.
2. Extract the **tools.jar** file found in the MelitaServer.jar using an extraction utility
3. Copy the **tools.jar** file in both
 - Java Runtime Environment/lib/ext/
 - Java Runtime Environment/jre/lib/ext/
4. Run the RMIRRegistry.exe which is located in Java Runtime Environment/bin/rmiregistry.exe
5. Run the Melita Server by typing

java -jar MelitaServer.jar

The last two steps must be repeated every time a user would like to run the Melita Server. The RMIRRegistry is the program responsible for interprocess communication between client and server.

Note 3 - Some suggestions

It is suggested that every time the server is run, the priority of the Server process is decreased. The reason is that since the Melita Server handles all the Adaptive Information Extraction functions it is the procedure which takes the most resources in terms of processing power and memory. Since this process does not need to run in real time and to avoid that precious processing power is taken away from the Melita client therefore making it unusable it is suggested that the user ...

1. opens the Task Manager (in windows)

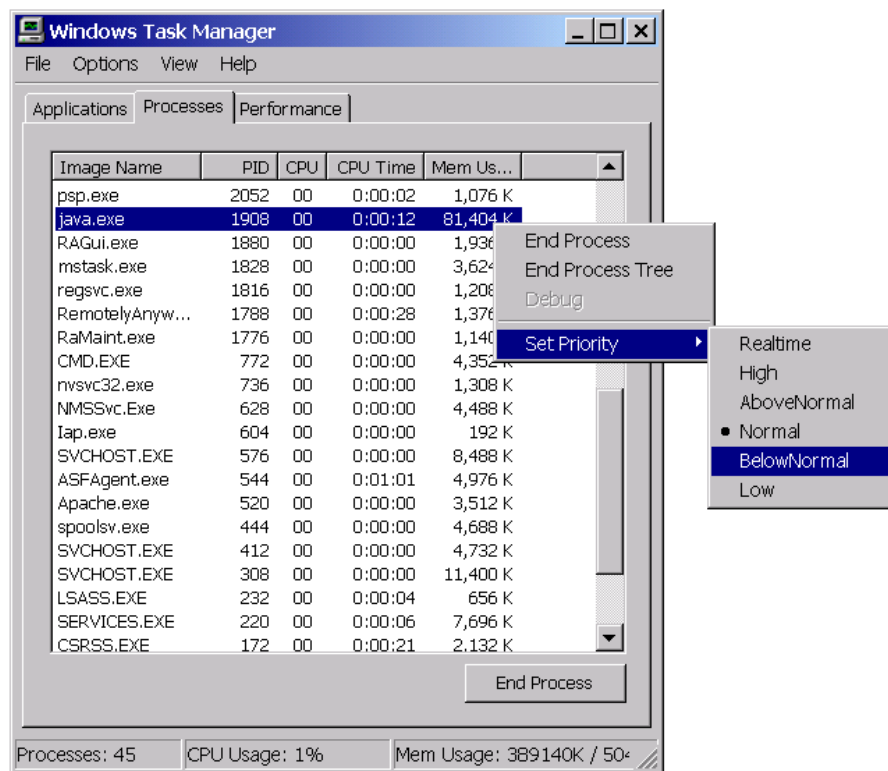


Figure 2.2: Setting the priority of the Melita Server in the Windows Task Manager.

2. selects the Melita Server process
3. right click on the process and a menu should popup
4. the user should go into `Set Priority -> Below Normal` and click the left mouse button (See Figure 2.2)
5. at this stage, a Windows message appears warning the user about the operation he just made and the user should press `OK`

A similar process exists of non-windows machines but goes beyond the scope of this document.

2.4 Running Melita for the first time

To run Melita for the first time there is nothing more to do than what was already described in the previous chapter i.e.

1. Double click on the Melita client.
2. Double click on the Melita server.

For non-windows systems, there's an additional step whereby the user must also launch the `RMIRegistry` as described in Section 2.3.2.

The order in which the client and the server are executed does not really matter because they are both independent of the other. If the client notices that the server is not available, the client simply does not make use of the server and vice versa. The application will still continue to run as normal. The only difference would be that if the server is not accessible by the client, no learning can be performed since all the learning is handled by the server.

Chapter 3

First steps in Melita

Taking a new step... is what people fear most.

- Dostoyevski

3.1 Setting a Scenario, as easy as ABC and D

Every time Melita is loaded, the user is presented with the Scenario Manager (See Figure 3.1). A scenario is a description of the domain being processed. The reason for having this scenario system is so that a user only sets the system once when the scenario is being created. All the subsequent uses of the system, the user only required to select the name of the session which he created earlier. At this stage, all the settings will be loaded in the Melita system automatically. The Scenario Manger is divided into three main sections. The session options, the details section and the session confirmation.

The session options lists all the different setup information required. In this case a session is made up of an Ontology (See Section 7.1 for more details), a corpus¹ of documents and some settings describing when the IE algorithm should intervene. In the next sections, we will look at each one of these in more detail. The options are represented using several tabs. It is worth nothing that every tab has a round circle followed by the name of the specific tab. If the colour of the circle is

white , it means that the information required by that particular section is missing.

green , it signifies that the system found all the information required and is correct.

¹A corpus can be thought of as a collection of texts gathered according to particular principles for some particular purpose.

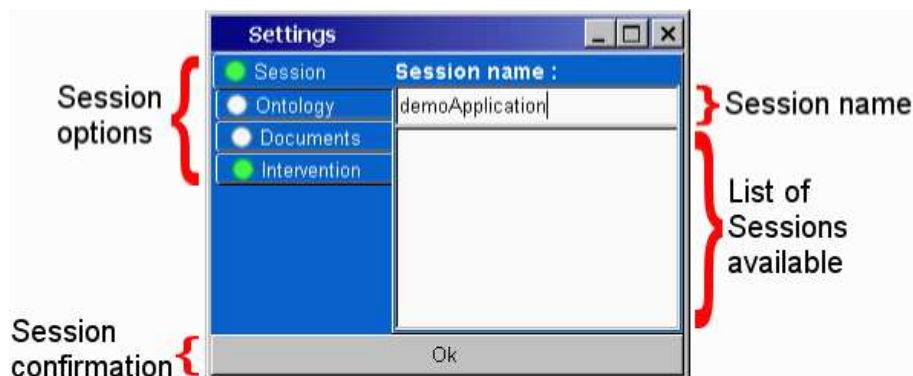


Figure 3.1: The Melita Scenario Manager.

red , it represents that even though there is some information available, the data is not correct and didn't pass all the verification procedures.

The details section changes according to the selected tab. It contains a form with all the information required to setup that particular section. In Figure 3.1 the main session tab is being displayed.

The session confirmation button at the bottom of the form is used to confirm all the settings entered by the user. It can be pressed only after all the buttons of the different tabs turn green. When the button is pressed a new session is created.

3.1.1 Assigning it a name ...

Every scenario is identified by a unique name. This name is selected by the user and describes the domain being processed. There are no restrictions imposed by the application on the name of the scenario. The only restrictions which exist are imposed by the file system being used since a scenario is stored in a file on disk. This section presents to the user a text box and a list box. The text box is used to enter a new session name and the list box lists all the existing sessions. If the user would like to load an existing session, he can select it from the list box and simply press the session confirmation button.

3.1.2 Browsing for an Ontology ...

The Ontology tab simply contains a text box and a **Browse** button. When pressed, the **Browse** button displays a dialogue which allows the user to navigate through the available files in order to locate the file containing the ontology. Melita accepts two kind of ontology files. One is an Amilcare² Scenario file

²Amilcare is the IE system used within Melita. It is a system for IE from Web documents for Knowledge Management that provides both accuracy and easy user customisation.

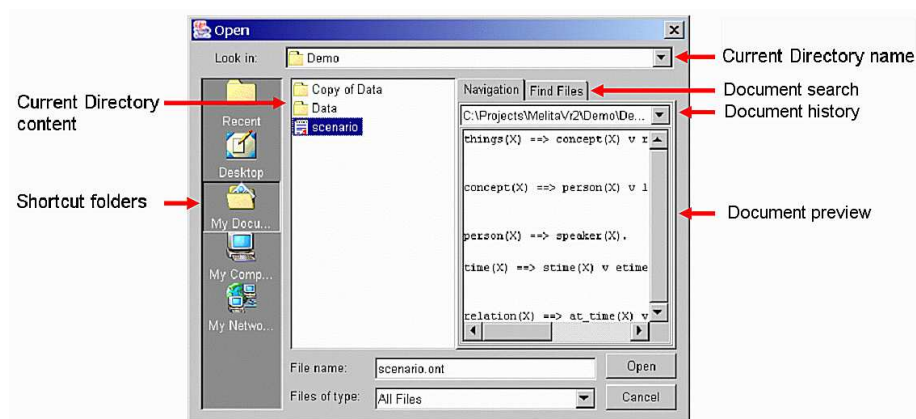


Figure 3.2: File selection.

(ontology.sce) and the other is a Melita Ontology (ontology.ont). The Melita Ontology is the preferred method of defining an ontology and is described in more details in Section 7.1.

The dialogue used in order to locate an ontology offers a number of advanced features (See Figure 3.2). It is similar to a normal file selection dialogue but it also has, a document history, a document preview and a document search incorporated in the same dialogue.

3.1.3 Choosing the Corpus ...

The Documents tab is made up of a list box together with an **Add** and **Remove** button. To add new documents to the corpus, the user must press the **Add** button and select either the documents or a directory containing the documents from a file chooser which pops up. In order to remove documents from the corpus, the documents are selected from the list box and the **Remove** button is pressed.

3.1.4 Dragging the intervention bar ...

The intervention bar is the bar that regulates when the IE system should intervene. For every concept in the Ontology, the system has a number of rules generated by the IE algorithm. Each rule is evaluated internally and a score between 1 and 100 is given to every rule. A score close to 100 shows a high level of precision while one nearer to 1 shows low precision. A user can instruct the system which rules to apply to the document. This is performed by adjusting the two knobs available in the intervention level interface (See Figure 3.3). One is called the **Certainty Level** and the other is the **Suggestion Level**. In the diagram, they are showing 75% and 25% respectively. The rules whose score

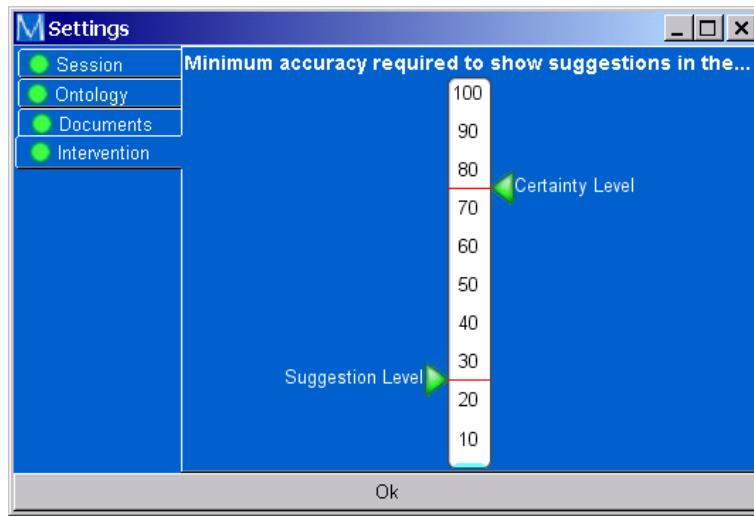


Figure 3.3: A user can adjust the global the intervention level by moving the two knobs.

goes above the **Certainty Level** will be shown to the user as annotations in the document whilst those between the **Certainty Level** and the **Suggestion Level** will be shown to the user as suggestions. The difference between the several kind of annotations in Melita will be explained later in Chapter 4.0.7. All rules below the **Suggestion Level** will not be shown to the user at all.

This bar in the Session Manager is a global one and controls all the different intervention levels of the sub-concepts in the Ontology. A change to this bar will result in changing all the other sub-concepts.

Chapter 4

The Melita Interface

Make everything as simple as possible, but not simpler.

- Albert Einstein

The Main Melita interface consists of 3 sections (See Figure 4.1), these are the Ontology panel, the Document panel and the Menus. The following sections will look in detail at each and everyone of these sections.

4.0.5 The Ontology panel

This section describes the Ontology viewer. The viewer renders the hierarchy found in the Ontology in the form of a tree representing concepts and relations. Every element in the tree is made up of four objects (See Figure 4.2). These are a check box, an small intervention graph, a label describing the type of element and the colour coded name of a concept or relation.

The check box is used to filter out concepts. Whenever it is clicked the instances of that particular concept are made **Visible** or **Invisible** in the Document panel. This feature is useful when the Ontology is very big and some colours may be reused for different concepts. In this case, to avoid confusing the user, the instances of a particular concept can be hidden by pressing this button.

The small intervention graph is similar to the intervention graph we saw before in Section 3.1.4. The only difference is that whilst the other one was global, this is a local one and its settings are only valid for this particular concept. The red line shows the certainty level while the green line shows the suggestion level. To change these levels, a user must press the mouse button on this intervention graph and a big intervention bar pops up in another window. The functionality is exactly like the one described in Section 3.1.4. Since this is a local intervention bar, whenever the knobs are adjusted, the user can see rules being applied to the document in the Document panel. These rules are applied

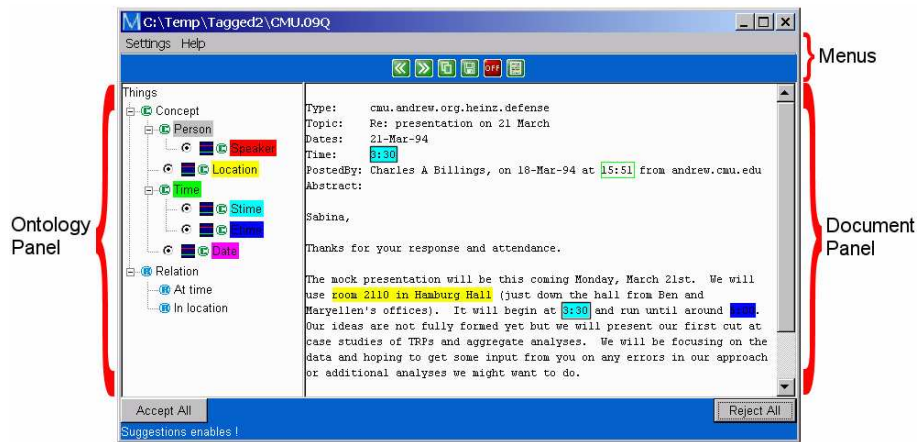


Figure 4.1: The Melita Main User Interface.



Figure 4.2: A concept in the Ontology panel.

in real time for that particular concept, this means that annotations will appear and disappear depending on which rules fire.

The label describing the type of element distinguishes between a concept and a relationship element. A concept element has a green © symbol while a relationship element has a blue ® symbol. This object is just for information purposes and has no other purpose.

The final object in the element is a name describing the particular concept with a colour associated to it. This is the colour which the annotation interface will use to mark instances of a particular concept. So if in Figure 4.1, the concept `Location` is marked with the colour yellow, the instance¹ of that concept in the document is also marked yellow. In order to select a concept from the Ontology, the user needs only double click on that concept and start annotating instances in the document panel.

The Ontology menu

The Ontology menu can be accessed by right clicking with the mouse on any concept in the Ontology. The menu is context sensitive and changes according

¹In this case the instance of the `Location` concept in Figure 4.1 is the following "room 2110 in Hamburg Hall"

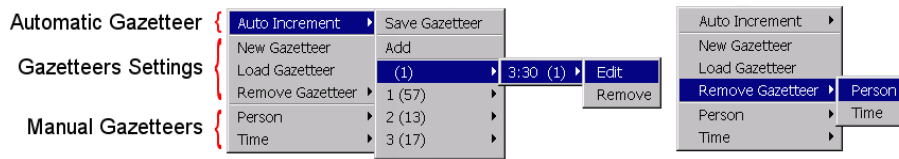


Figure 4.3: The menu which pops up when the user right clicks on a concept in the Ontology.

to the concept being selected. All the options available help the user to create a gazetteer² for that particular concept. The gazetteer is very powerful because apart from a list of instances, the user can also define regular expressions³.

This menu is divided into three main sections, these are the Automatic Gazetteers, the Gazetteers Settings and the Manual Gazetteers. The whole menu structure can be seen in Figure 4.3.

The Gazetteer settings allows the user to create a New Gazetteer, Load an existing Gazetteer and Remove a loaded Gazetteer. To create a new Manual Gazetteer, the user needs to only specify the name of the gazetteer. In order to populate it with instances, this will be shown in the coming paragraphs. To load an existing gazetteer⁴, a file open dialogue is displayed and the user is asked to select a gazetteer file from disk. Removing a gazetteer is simply a matter of selecting the gazetteer to remove from the list provided.

The Automatic Gazetteer is a gazetteer generated by the system at start-up. It includes all the instances already tagged by the user and is generated by going through the previously annotated document and gathering the list of instances for every concept. Every gazetteer submenu has the following structure, a Save option, an Add option and a list of all the instances available. The Save option for the automatic gazetteer saves a copy of the gazetteer to disk and also adds the same copy to the Manual Gazetteers. The main reason for doing this is that even though the Automatic gazetteer lists all the instances available in the document, it does not apply the examples it possesses back to the text. This is done because this gazetteer is generated automatically and therefore any changes made to this gazetteer by the user will be lost unless they are saved in a manual gazetteer. Therefore it is imperative that if the user would like to make use of this gazetteer, he must save a copy and modify the copy in the Manual gazetteer's list. Any number of gazetteers can be associated with the same concept. The next option allows the user to Add an element in the gazetteer by using the Gazetteer editor.

The Gazetteer editor (See Figure 4.4) is made up of three boxes at the top, a

²A gazetteer is a file containing a list of names of elements which belong to a particular class of concepts.

³A regular expression is a string that describes a whole set of strings, according to certain syntax rules. These expressions are used by many text editors and utilities to search bodies of text for certain patterns.

⁴To learn how to create gazetteers for Melita refer to Section 10.1.

Pre-Filler	Filler	Post-Filler
1	3:30	
2 Time:	3:30	
3 Time:	\d+:\d+	
4 Time:	\d+:\d+\W*?[PpAa]\.?[Mm]\.?	

Table 4.1: Examples of regular expressions which can be used to match the time "3:30"

list and three buttons at the bottom. The three text boxes represent (from left to right) a pre-filler, a filler and a post-filler. Lets try to understand how this work by look at the example in Table 4.1. The filler is the string we would like to find, in the case of the example, its a time (3:30). The pre-filler is the text that comes before the filer. It is no use to us but can help us locate the string we would like to find. In the example, time is normally preceded by the string "Time:". The post-filler is similar to the pre-filler except that it comes after the filler and not before. So using these three boxes we can model the information we would like to find. To use this editor one needs not learn anything new, but if the user would like to exploit the power offered by the editor, it would be an asset to learn how to use regular expressions (See Chapter 8.1). If we take a look once again at the example, we notice that the simplest pattern is to use a string which matches exactly the string we would like to find. This obviously works but matches all the occurrences of the string which we enter. Obviously this includes some strings which might not be relevant to our search. To refine a little our search, we can take a look at example 2 and include some context as well (like the string "Time:" which appears before the string we would like to find most of the times). The third example, generalises over the previous one and introduces a regular example pattern used to match more examples and not just the string "3:30". The pattern just says, *one or more digit, followed by a ":"*, *followed by one or more digits*. So this patter also matches things like "Time: 2:30", "Time: 12:30", "Time: 7:00", etc. But it might be useful to capture also whether the time is in the morning or in the afternoon. This is done by refining further the pattern like in example 4. This pattern has the first part equivalent to example 3 but adds after it, *zero or more characters which are not digits or letters, followed by one of "P" or "p" or "A" or "a", followed by one or no occurrences of a dot, followed by one of "M" or "m", followed by one or no occurrences of a dot*. It is quite simple to realise that this pattern has results similar to the previous patter but adds also another restriction that it must also have an "AM" or a "PM" after it. So this patter matches things like "Time: 2:30 pm", "Time: 12:30 P.m.", "Time: 7:00 AM", etc. It is worth noting that the pre-filler, filler and post-filler all can have either a string or a regular expression pattern defined in them or even none (as in the example of post-filler). Obviously a filler with nothing inside it is not much useful though!

Back to the gazetteer editor, once the user is happy with the pattern, he can test it in real time by pressing the test button. This immediately presents

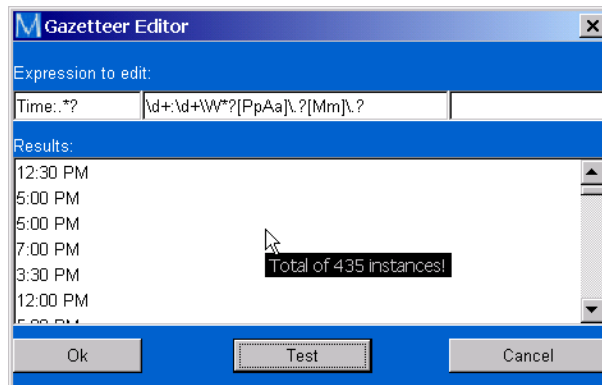


Figure 4.4: The Gazetteer editor.

to the user in the list, all the instances where that pattern matches. The user can move the mouse on the list box and a tool tip will pop-up showing to the user the number of occurrences of that pattern. The user can also navigate to the document where the particular instance is found just by pressing one of the instances in the list box. If the user is happy, the pattern is added to the gazetteer by pressing the "OK" button, otherwise, the "Cancel" button can be pressed to cancel everything.

If we continue our tour of the gazetteer menu, we reach the list of all the instances available. This list contains all the instances sorted in alphabetical order and divided into several submenus (according to the first letter of the instance) for easier viewing (See Figure 4.3). This division is performed because the amount of instances can grow, and it might not be visible in just one menu. For every instance, the user can see the number of occurrences (displayed in brackets) of that pattern in all the corpus. The instance menu offers two options, one to edit the pattern and the other to remove the pattern. The edit option opens a Gazetteer editor in which the user can edit the pattern in the same way as it was shown before. The instances shown to the user are context sensitive and change according to the underlying concept chosen. All the submenus of the Manual gazetteers operate in the same way like this menu. When the user finishes editing the gazetteers, the new changes will become visible as soon as a new document is displayed in the Document panel.

4.0.6 Editing NLP Rules

It is possible to inspect and modify the rules induced by the underlying IE engine by selecting the item 'NLP Rules' in the ontology menu. The IE engine currently used in Melita is Amilcare (<http://nlp.shef.ac.uk/amilcare>). A window with the list of induced NLP rules for the selected tag is shown (See Figure 4.5).

For each rule the following information is presented:

Num	Tag	Pattern	Matches	Score
0	<date>	(LexCatcd) (Sem<month>) (Ne<date>) <DATE>	70	1
1	<date>	(LexCatcd) (Sem<month>) (LexCatcd) <DATE>	71	1
2	<date>	(Sem<male>) 0 0 (Sem<month>) (Ne<date>) <DATE>	15	1
3	<date>	(Sem<male>) 0 0 0 (LexCatcd) <DATE>	20	1
4	<date>	(Sem<male>) 0 0 (Sem<month>) (LexCatcd) <DATE>	15	1
5	<date>	(Tok2002) <DATE> (Capupperinitial)	17	1
6	<date>	(Tok2002) <DATE> (LexCatnnp)	15	1
7	<date>	(Ne<date>) <DATE> (LexCatnnp) 0 0 0 (LexCatnnp)	20	1
8	<date>	(Ne<date>) <DATE> (LexCatnnp) 0 (Caplowercase)	15	1.001
9	<date>	(Tok2002) <DATE>	9	1.001
10	<date>	(Tokapr) (Tok2002) <DATE>	4	1.007
11	<date>	(Tokmar) (Tok2002) <DATE>	2	1.05
12	<date>	(Toknovember) (Tok2002) <DATE>	1	1.155
13	<date>	(Tokaug) (Tok2002) <DATE>	1	1.155
14	<date>	<DATE> (Semnumber) (Semmonth)	72	1
15	<date>	<DATE> (LexCatcd) (Semmonth)	72	1
16	<date>	(Sem<male>) (Tok) <DATE> (Semnumber)	15	1
17	<date>	(Sem<male>) (LexCat) <DATE> (Semnumber)	15	1
18	<date>	<DATE> (Semnumber) 0 (Tok2002) (Capupperinitial)	17	1
19	<date>	<DATE> (LexCatcd) 0 (Tok2002) (Capupperinitial)	17	1
20	<date>	<DATE> (Semnumber) 0 (Tok2002) (LexCatnnp)	11	1
21	<date>	<DATE> (LexCatcd) 0 (Tok2002) (LexCatnnp)	11	1
22	<date>	<DATE> (Semnumber) (Semambig) (LexCatnnp)	11	1
23	<date>	<DATE> (Semnumber) (Semambig)	10	1
24	<date>	<DATE> (LexCatcd) (Semambig)	10	1
25	<date>	(LexCat) (Sem<male>) 0 (Tok) <DATE>	21	1

Figure 4.5: NLP rules per tag.

1. Rule number
2. Tag
3. Pattern
4. Matches: the number of times a rule was fired on the current corpus
5. Score: the score (typically number of errors/number of fires) that the rule has if the corpus is tagged with the correct solutions (otherwise it is 1.0)

A pattern is a set of conditions on a sequence of words. Each word is represented in parentheses. A word in the text presents a number of features potentially tested by the rules:

1. Tok (original token cast in lowercase)
2. Lemma (if a lemmatizer is provided)
3. Pos: part of speech (e.g. noun). The brill's tagset is used.
4. Sem: a tag as provided by the gazetteer or a user defined dictionary
5. Cap: capitalization (e.g. uppercase)
6. Nerc: a tag returned by Annie's named entity recogniser
7. Oth: typically html tags found in the text.

It is possible to modify the induced rules either by editing, copying or deleting. Select the rule you want to work on and select the corresponding button

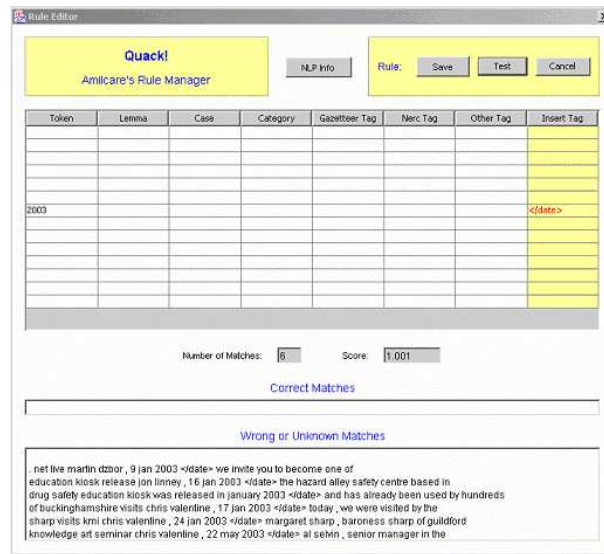


Figure 4.6: The editor.

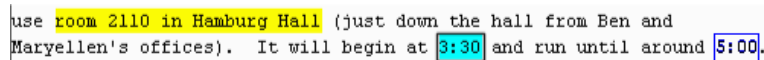
(edit, copy - in both cases the editor is open— and delete). Double-clicking on a rule opens the editor as well. The editor appears as in Figure 4.6.

Some of the fields can be filled by using some predefined menus. The menu is generated using the values found in the corpus. In the middle and low panels, the details of the matches are shown, divided with wrong and correct. If the corpus was not tagged all the matches are shown as incorrect. A window in the texts is shown with the annotation inserted by the rule. It is possible to test the rule before saving it and to see the cases matched and compute the score. When a rule is saved, it becomes available to the underlying IE system.

4.0.7 The Document panel

The Document panel is the place where the documents are displayed and annotated by the user. This panel works in a very simple way. In fact it is just a matter of clicking on a concept in the Ontology and marking the concept in the document. This is done by clicking on the word in the document and dragging until all the word or a group of words are highlighted. As simple as that! On thing to note is that the left and right mouse buttons have different effects. The left mouse button highlights word by word while the right mouse button highlights letter by letter. This means that with the left mouse button you can only mark whole words or groups of them. With the right mouse button one can mark parts of a world.

Another important thing to know about the document panel is the different kinds of annotations possible. There are three different kind of annotations. The



use room 2110 in Hamburg Hall (just down the hall from Ben and Maryellen's offices). It will begin at 3:30 and run until around 5:00.

Figure 4.7: Example of the different kind of annotations.

first kind represent all those annotations which are inserted by the user. They have the same colour as the respective concept in the ontology and are shown as a coloured rectangular box which can span multiple lines (See the yellow annotation in Figure 4.7). The second kind represents all the suggestions given by the system. These include suggestions given by the learning algorithm and those by the gazetteers. These annotations are shown as a white rectangular box with a coloured border (having the same colour as the respective concept in the ontology) which can span multiple lines (See the dark blue annotation in Figure 4.7). The last kind of annotation represents the certainties⁵ given by the learning algorithm. These annotations are similar to the user's annotations except for the fact that they have a black border around them (See the light blue annotation with a black border in Figure 4.7).

To delete an annotation from the document, the user must double click on the annotation and it just disappears. These annotations are stored in the document as XML markups⁶. The document panel also has two buttons beneath it and a status bar. The two buttons allow the user to **Accept All** the suggestions in the current document or to **Remove All** of them. The remove button is a little bit tricky because it has a dual function. If the document contains suggestions when pressed, then the suggestions are removed and all the other tags remain intact. If there are no more suggestions but only tags, then the tags are removed. The status bar displays information to the user about Melita and also about the program's interaction with the server.

4.0.8 The Menus

Melita has two types of menus. A pull down menu at the top and a tool bar menu in the middle of the interface made up of six buttons.

The pull down menu is made up of two main menus, the settings and the Help menu. The Settings menu allows the user to change the scenario settings like the current session, the ontology, the corpus and the global intervention level. The Help menu pops up an window with information about the Melita system, licensing, etc.

The tool bar menu has six buttons in total. These are:

Previous button - shows the previous document in the document panel.

Next button - shows the next document in the document panel.

⁵These are those annotations generated by those rules whose rating is higher than the certainty level.

⁶An annotation around the time 3:30, will look like `<time>3:30</time>` in the document.

Documents button - shows the list of documents.

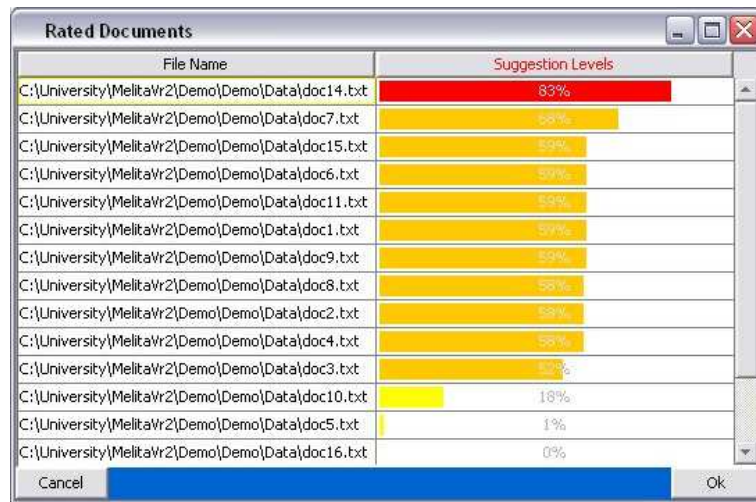
Save button - saves all the annotations inserted in the current document.

IE button - enable and disable the IE engine.

Suggestion button - displays suggestions obtained from the IE engine in the document.

Note 1

The documents button allows the user to navigate through the documents using no particular order. A user can just select a document from the list and press "Ok". This document will be the current document in the document panel. If cancel is pressed, nothing is changed. Next to the document list there's another column called the **Suggestion Levels**. These levels show which documents will benefit most the learning algorithm if they are annotated. The documents with the highest % are those which the learning algorithm did not manage to find any examples. Therefore if a user annotates these documents, it is guaranteed that new patterns will be discovered. For further information about this smart annotation of documents, please refer to Chapter 11.1; The documents in the list can be ordered in two ways, either by name or by the % in the suggestion levels. This is changed by clicking on the title of the table.



File Name	Suggestion Levels
C:\University\MelitaWr2\Demo\Demo\Data\doc14.txt	83%
C:\University\MelitaWr2\Demo\Demo\Data\doc7.txt	58%
C:\University\MelitaWr2\Demo\Demo\Data\doc15.txt	59%
C:\University\MelitaWr2\Demo\Demo\Data\doc6.txt	59%
C:\University\MelitaWr2\Demo\Demo\Data\doc11.txt	59%
C:\University\MelitaWr2\Demo\Demo\Data\doc1.txt	59%
C:\University\MelitaWr2\Demo\Demo\Data\doc9.txt	59%
C:\University\MelitaWr2\Demo\Demo\Data\doc8.txt	58%
C:\University\MelitaWr2\Demo\Demo\Data\doc2.txt	58%
C:\University\MelitaWr2\Demo\Demo\Data\doc4.txt	58%
C:\University\MelitaWr2\Demo\Demo\Data\doc3.txt	53%
C:\University\MelitaWr2\Demo\Demo\Data\doc10.txt	18%
C:\University\MelitaWr2\Demo\Demo\Data\doc5.txt	1%
C:\University\MelitaWr2\Demo\Demo\Data\doc16.txt	0%

Figure 4.8: Documents rankings in Melita

Note 2

When browsing through the documents the suggestion button changes colour from a green background to a red background. This means that there are

suggestions from the IE for that particular document. If the button is pressed, these suggestions are added to the current document.

Chapter 5

Contact details

Flatter me, and I may not believe you. Criticize me, and I may not like you. Ignore me, and I may not forgive you. Encourage me, and I will not forget you.

- William Arthur Ward

If for any reason you would like to contact the people responsible with the development of Melita, you can do so by ...

sending an E-mail to Melita@dcs.shef.ac.uk

giving us a ring on +44 (0)114 222 1814

sending us a fax on +44 (0)114 222 1810

visiting us or writing a letter to The Natural Language Processing Group,
Department of Computer Science,
University of Sheffield,
Regent Court 211,
Portobello Street Sheffield S1 4DP UK

... and ask for Mr Alexiei Dingli, Dr Fabio Ciravegna or Mr Jose Iria.

Chapter 6

Conclusion

To finish a work? To finish a picture? What nonsense! To finish it means to be through with it, to kill it, to rid it of its soul, to give it its final blow the coup de grace for the painter as well as for the picture.

- Pablo Picasso

Unfortunately everything comes to an end and so does this document. This is not an ugly moment though! Not cause i didn't enjoy writing this document but because i don't like to think at the end as finishing something. I like to think of the end as the beginning of new stuff, after all sometimes it is imperative for things to finish in order to regenerate other things. This document tried to give a comprehensive overview of the Melita tool. It tried to answer questions like *What it is?*, *How to use it?* and lots of other stuff. Obviously it is far from being perfect and knowing a little human nature (being one myself) the human mind will surely come up with a zillion things i haven't thought of and which would be useful to add in this document. If you have any of these, just contact me and i'll see if i can add them to the next revision of this document. Apart from that, i don't have anything to add except for ...

Have fun using Melita !

Chapter 7

Index A

7.1 The Ontology

7.1.1 What is an Ontology?

An ontology is a specification of a conceptualization. This means that an ontology is a description of the concepts and relationships that can exist between objects in a system. What is important is what an ontology is for. Ontologies in computer science are normally used to enable knowledge sharing and reuse. Although this is not the only way to specify concepts and relationships between them, it has some nice properties for knowledge sharing among AI software, such as the fact that commitment to use an ontology can be seen as an agreement to use a vocabulary in a way that is consistent (but not complete) with respect to the theory specified by an ontology. Ontologies are built so that both automated systems and people manage to share knowledge with and among themselves using a uniform definition of the micro-world they are operating upon. A commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology.

7.1.2 Ontology creation for dummies

In order to start creating an Ontology, there are a number of steps one must take. First of all, the possible object types in a micro-world must be identified together with the possible relationships between them. Once this is done, its just a matter of expressing the Ontology in the syntax understood by Melita. This syntax is based around the XI language (See [Gaizauskas and Humphreys, 1996]) a prolog based language. Lets look at an example ...

Imagine we would like to model the seminar announcements domain. A seminar announcement is assumed to correspond to a record in a relational database, containing the fields: name of the speaker, location of the seminar, and the start and end times of the seminar. Any field may or may not be

instantiated for a given announcement. If it is instantiated, it takes a single text fragment.

So, our concepts can be either a **Person**, a **Location** or a **Time**. The concept **Person** has a specialisation called a **Speaker**, while the concept **Time** has two sub-concepts which are **Start time** and **End time**. A time has a relation called **At time** and a location has a relation called **In location**.

Lets see how this is represented in Melita. First there is a top level concept which must be in all Ontologies which is the concept **things** and **things** contain either a number of concepts or a number of relations. This is expressed in the following syntax.

$$things(X) ==> concept(X) v relation(X).$$

Now underneath the concept, we can start defining our hierarchy. Therefore to say that a concept can be either a parson, a location or a time, we write it as follows:

$$concept(X) ==> person(X) v location(X) v time(X).$$

But a person can be a speaker, this is written as ...

$$person(X) ==> speaker(X).$$

And a time can either be a start time or an end time, written as ...

$$time(X) ==> stime(X) v etime(X).$$

Finally to represent the two relations **at time** and **in location** we use ...

$$relation(X) ==> at_time(X) v in_location(X).$$

That's all, please note some small things. Spaces and other special characters are not allowed in the names of concepts or relations. Relations are not used in any way by Melita so they can be ignored although it is suggested that you write them for completeness sake. The underscore (`_`) in the names of the relations has a special meaning for Melita, it means that the user can't annotate instances of this class. Finally all this is stored in a file with a `.ont` extension after the file name. The following is the contents of the **SeminarOntology.ont** file ...

<pre> things(X) ==> concept(X) v relation(X). concept(X) ==> person(X) v location(X) v time(X). person(X) ==> speaker(X). time(X) ==> stime(X) v etime(X). relation(X) ==> at_time(X) v in_location(X). </pre>

Chapter 8

Index B

8.1 Introduction to Regular Expressions

This section¹ is intended to give a brief introduction to regular expressions. It is not intended to be a comprehensive Bible for regular expressions. Further information including more comprehensive tutorials with examples can be found on the web. Unless you're an avid regular expressions² user, the initial regex jargon might confuse you. What are quantifiers and the differences among greedy, reluctant, and possessive quantifiers? What are character classes, boundary matchers, back references, and embedded flag expressions? To answer those and other questions, we explore many of the regex constructs, or regex pattern categories, that Pattern recognizes. We begin with the simplest regex construct: literal strings.

8.1.1 Literal strings

You specify the literal string regex construct whenever you type a literal string in the text field of the gazetteer editor. So if you specify "3:30" as a literal string regex construct that consists of literal characters 3, :, 3, and 0 (in that order).

8.1.2 Metacharacters

Although literal string regex constructs are useful, more powerful regex constructs combine literal characters with metacharacters. For example, in a.b, the period metacharacter (.) represents any character that appears between a and b. To see the period metacharacter in action, if we consider the pattern ".ox" on the string "The quick brown fox jumps over the lazy ox.". The regex system searches the text for matches that begin with any character and end with

¹adapted from a tutorial found at <http://www.javaworld.com/javaworld/jw-02-2003/jw-0207-java101-p2.html>

²also called regex.

ox, and produces the following matches "fox" and " ox". The . metacharacter matches the f in the first match and the space character in the second match.

Tip

To specify . or any metacharacter as a literal character in a regex construct, quoteconvert from meta status to literal status the metacharacter in one of two ways:

- Precede the metacharacter with a backslash character.
- Place the metacharacter between $\backslash Q$ and $\backslash E$ (e.g., $\backslash Q.\backslash E$).

8.1.3 Character classes

We sometimes limit those characters that produce matches to a specific set of characters. For example, we might search text for vowels a, e, i, o, and u, where any occurrence of any vowel indicates a match. A character class, a regex construct that identifies a set of characters between open and close square bracket metacharacters ([]), helps us accomplish that task. Pattern supports the following character classes:

Simple: consists of characters placed side by side and matches only those characters. Example: [abc] matches characters a, b, and c.

Negation: begins with the ^ metacharacter and matches only those characters not in that class. Example: [^abc] matches all characters except a, b, and c.

Range: consists of all characters beginning with the character on the left of a hyphen metacharacter (-) and ending with the character on the right of the hyphen metacharacter, matching only those characters in that range. Example: [a-z] matches all lowercase alphabetic characters.

Union: consists of multiple nested character classes and matches all characters that belong to the resulting union. Example: [a-d[m-p]] matches characters a through d and m through p.

Intersection: consists of characters common to all nested classes and matches only common characters. Example: [a-z&&[d-f]] matches characters d, e, and f.

Subtraction: consists of all characters except for those indicated in nested negation character classes and matches the remaining characters. Example: [a-z&&[^m-p]] matches characters a through l and q through z.

Predefined character class	Description
<code>\d</code>	A digit. Equivalent to <code>[0-9]</code> .
<code>\D</code>	A nondigit. Equivalent to <code>[^0-9]</code> .
<code>\s</code>	A whitespace character. Equivalent to <code>[\t\n\x0B\f\r]</code> .
<code>\S</code>	A nonwhitespace character. Equivalent to <code>[^\s]</code> .
<code>\w</code>	A word character. Equivalent to <code>[a-zA-Z_0-9]</code> .
<code>\W</code>	A nonword character. Equivalent to <code>[^\w]</code> .

Table 8.1: Predefined character classes

Tip

Combine multiple ranges within the same range character class by placing them side by side. Example: `[a-zA-Z]` matches all lowercase and uppercase alphabetic characters.

8.1.4 Predefined character classes

Some character classes occur often enough in regexes to warrant shortcuts. Pattern provides such shortcuts with predefined character classes, which Table 8.1 presents. Use predefined character classes to simplify your regexes and minimize regex syntax errors.

8.1.5 Capturing groups

Pattern supports a regex construct called a capturing group that saves a match's characters for later recall during pattern matching; that construct is a character sequence surrounded by parentheses metacharacters `()`. All characters within that capturing group are treated as a single unit during pattern matching. For example, the `(Java)` capturing group combines letters J, a, v, and a into a single unit. This capturing group matches the Java pattern against all occurrences of Java in text. Each match replaces the previous match's saved Java characters with the next match's Java characters. Capturing groups can nest inside other capturing groups. For example, in `(Java(language))`, `(language)` nests inside `(Java)`.

8.1.6 Quantifiers

Quantifiers are probably the most confusing regex constructs to understand. Part of that confusion comes from trying to grasp Pattern's 18 quantifier categories (organized as three major categories of six fundamental quantifier categories). Another part of that confusion comes from trying to decipher the concept of zero-length matches. Once you understand that concept and those 18 categories, much (if not all) of the confusion disappears.

A quantifier is a regex construct that implicitly or explicitly binds a numeric value to a pattern. That numeric value determines how many times to match a pattern. Pattern's six fundamental quantifiers match a pattern ...

? once or not at all

* zero or more times

+ one or more times

{x} an exact x number of times

{x,} at least x times

{x,y} at least x times but no more than y times

The six fundamental quantifier categories replicate in each of three major categories: greedy, reluctant, and possessive. Greedy quantifiers attempt to find the longest match. In contrast, reluctant quantifiers attempt to find the shortest match. Possessive quantifiers also try to find the longest match. However, they differ from greedy quantifiers in how they work. Although greedy and possessive quantifiers force a matcher to read in the entire text prior to attempting a first match, greedy quantifiers often cause a matcher to make multiple attempts to find a match, whereas possessive quantifiers cause a matcher to attempt a match only once.

The following examples on the string "abaa" illustrate the behavior of the six fundamental quantifiers in the greedy category, and the behavior of a single fundamental quantifier in each of the reluctant and possessive categories. These examples also introduce the zero-length match concept:

a? matches zero or one time, therefore we have **abaa**, **abaa**, **abaa**, **abaa** and **abaa**. The output reveals five matches. Although the first, third, and fourth matches come as no surprise in that they reveal the positions of the three **a**s in **abaa**, the second and fifth matches are probably surprising. Those matches seem to indicate that **a** matches **b** and also the text's end. However, that is not the case. **a?** does not look for **b** or the text's end. Instead, it looks for either the presence or lack of **a**. When **a?** fails to find **a**, it reports that fact as a zero-length match, a match of zero length where the start and end indexes are the same. Zero-length matches occur in empty text, after the last text character, or between any two text characters.

a* matches zero or more times, therefore we have **abaa**, **abaa**, **abaa** and **abaa**. The output reveals four matches. As with **a?**, **a*** produces zero-length matches. The third match, where **a*** matches **aa**, is interesting. Unlike **a?**, **a*** matches either no **a** or all consecutive **a**s.

a+ matches one or more times, therefore we have **abaa** and **abaa**. The output reveals two matches. Unlike **a?** and **a***, **a+** does not match the absence of

Boundary Matcher	Description
<code>^</code>	The beginning of a line.
<code>\$</code>	The end of a line.
<code>\b</code>	A word boundary.
<code>\B</code>	A non-word boundary.
<code>\A</code>	The beginning of the text.
<code>\G</code>	The end of the previous match.
<code>\Z</code>	The end of the text (but for the final line terminator, if any).
<code>\z</code>	The end of the text.

Table 8.2: Boundary matchers

a. Thus, no zero-length matches result. Like `a*`, `a+` matches all consecutive `as`.

`a+?` matches one or more times but using a reluctant quantifier, therefore we have `abaa`, `abaa` and `abaa`. Unlike its greedy variant in the third example, the reluctant example produces three matches of a single `a` because the reluctant quantifier tries to find the shortest match.

`a{2}` matches two times exactly, therefore we have just `abaa`.

`a{1,}` matches at least one time, therefore we have `abaa` and `abaa`.

`a{1,2}` matches at least one time and at most two times, therefore we have `abaa`, `abaa` and `abaa`.

8.1.7 Boundary matchers

We sometimes want to match patterns at the beginning of lines, at word boundaries, at the end of text, and so on. Accomplish that task with a boundary matcher, a regex construct that identifies a match location. Table 8.2 presents Pattern’s supported boundary matchers. The following example uses the `^` boundary matcher metacharacter to ensure that a line begins with “The” followed by zero or more word characters:

If we use the pattern `^The\w*` on the word “Therefore” we find that `^` indicates that the first three text characters must match the pattern’s subsequent `T`, `h`, and `e` characters. Any number of word characters may follow. The pattern manages to match the word “Therefore”. If the input word is changed to “ Therefore”. Using the same pattern, no match is found because a space character precedes “Therefore”.

Chapter 9

Appendix C

9.1 Part-of-Speech Tags used in the Hepple Tagger

NN	noun singular or mass
NNP	proper noun - singular
NNPS	proper noun - plural
NNS	noun - plural
NP	proper noun - singular
NPS	proper noun - plural
JJ	adjective
JJR	adjective - comparative
JJS	adjective - superlative
JJSS	-unknown-, but probably a variant of JJS
RB	adverb

RBR	adverb - comparative
RBS	adverb - superlative
VB	verb - base form
VBD	verb - past tense
VBG	verb - gerund or present participle
VBN	verb - past participle
VBP	verb - non-3rd person singular present
VBZ	verb - 3rd person singular present
FW	foreign word
CD	cardinal number
CC	coordinating conjunction
DT	determiner
EX	existential 'there'
IN	preposition or subordinating conjunction
LS	list item marker
MD	modal
PDT	predeterminer
POS	possessive ending
PP	personal pronoun
PRP	-unknown-, but probably possessive pronoun
PRP\$	-unknown-, but probably possessive pronoun
PRPR\$	-unknown-, but probably possessive pronoun
RP	particle
TO	literal "to"
UH	interjection
WDT	'wh'-determiner
WP	'wh'-pronoun
WP\$	possessive 'wh'-pronoun
WRB	'wh'-adverb
SYM	symbol
"	literal double quotes
#	literal pound sign
\$	literal dollar sign
'	literal single quote or apostrophe
(literal left parenthesis
)	literal right parenthesis
,	literal comma
-	literal double-dash
-LRB-	-unknown-
.	literal period
::	literal colon
'	literal grave
STAART	start state marker (used internally)

Chapter 10

Index D

10.1 The Gazetteer File structure

Any user can add his own gazetteer. This section will explain the main structure required by a Melita gazetteer. There are two main requirements, first of all, the gazetteer must be an XML file and secondly it must have the following structure ...

`<xml version="Melita">` - Start XML tag. The tag also contains a version attribute. For a Melita Gazetteer to be valid, the version must *always* be "Melita".

`<concept name="location">` - Tag starting the list of elements associated with a concept. The name attribute indicates the name of the concept and must be equal to the name of the concept in the Ontology. In this case the concept is called "location". A Melita gazetteer can have an unlimited number of concepts specified in the same file.

`<element occurrence="5">WeH 6121</element>`

`<element occurrence="2">Baker Hall 235A</element>` - Every concept in the gazetteer must have a number of instances. These are defined using the element tag. In this case, the concept location has two instances "WeH 6121" and "Baker Hall 235A". This tag has an attribute occurrence which states the number of times that instance appears in the corpus of documents. This attribute is used only for statistical purposes and it can be ignored, but it can not be omitted from the element tag. If unsure, always set this attribute to "1".

`</concept>`

`</xml>`

Chapter 11

Index E

11.1 Theory behind Melita

In order to have a better understanding how Melita works and the theory behind it, please refer to the following documents:

- **Timely and Non-Intrusive Active Document Annotation via Adaptive Information Extraction** [Ciravegna et al., 2002a]
- **User-System Cooperation in Document Annotation based on Information Extraction** [Ciravegna et al., 2002b]
- **Using Adaptive Information Extraction for Effective Human-centred Document Annotation** [Ciravegna et al., 2003]

Bibliography

- F. Ciravegna, A. Dingli, D. Petrelli, and Y. Wilks. Timely and non-intrusive active document annotation via adaptive information extraction. In *Semantic Authoring, Annotation and Knowledge Markup (SAAKM02)*. ECAI, 2002a.
- F. Ciravegna, A. Dingli, D. Petrelli, and Y. Wilks. User-system cooperation in document annotation based on information extraction. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management, EKAW02*. Springer Verlag, 2002b.
- F. Ciravegna, A. Dingli, Y. Wilks, and D. Petrelli. Using adaptive information extraction for effective human-centred document annotation. In *Text Mining, Theoretical Aspects and Applications*. Springer Verlag, 2003.
- H. Cunningham, D. Maynard, V. Tablan, C. Ursu, and K. Bontcheva. "developing language processing components with GATE", 2002. www.gate.ac.uk.
- R. Gaizauskas and K. Humphreys. Xi: A simple prolog-based language for cross-classification and inheritance. In *7th International Conference on Artificial Intelligence*, 1996.
- S. Handschuh, S. Staab, and F. Ciravegna. S-cream — semi-automatic creation of metadata. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, October 2002.
- K. Hook. Steps to take before intelligent user interfaces become real, 2000. URL citeseer.nj.nec.com/440860.html.
- M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. Mnm: Ontology driven semi-automatic and automatic support for semantic markup, 2002. URL citeseer.nj.nec.com/545549.html.