Seventh Framework Programme
Theme 3:
Information and Communication Technologies

Challenge 6:
ICT for Safety and Energy Efficiency in Mobility
Grant Agreement Number 318452

Personalized Mobility Services for energy efficiency and security through advanced Artificial Intelligence Techniques

Deliverable D4.2

Knowledge capture services

Work package 4
Reasoning and decision function

Leading Partner: JSI

Security Restriction: PU (Public)

21/10/2013

Version 1.0

Versioning

Version	Description	Author	Date	Comments
0.1	First circulating draft version	Luka Bradesko	23/09/2013	Structure of deliverable and requesting inputs to each section
0.1	Added draft of Knowledge Capture section	Paulo Figueiras	23/09/2013	Added draft of Knowledge Capture section
0.2	Natural Knowledge Elicitation	Luka Bradesko	24/09/2013	
0.2	Updated Knowledge Capture section	Paulo Figueiras	24/09/2013	
0.3	KA in traffic centers	Zala Herga	26/09/2013	
0.4	Pattern Based KA	Janez Starc	27/09/2013	
0.5	Updated Natural KA and integration	Luka Bradesko	27/09/2013	
0.6	Conclusions and summary	Luka Bradesko	30/09/2013	

Contributors

Name	Organization	Notes
Luka Bradesko	JSI	
Paulo Figueiras	UNINOVA	
Zala Herga	JSI	
Janez Starc	JSI	

Annexes

No	File name	Title
1	CC_KnowledgeAcquisition.pdf	Call Centre Knowledge Acquisition And Decision Support Prototype
2	PersonalizedTravelCompanion.pdf	Knowledge Base Approach for Developing a Mobile Personalized Travel Companion

Table of Contents

Table of Figures

1 Executive summary

This deliverable presents the directions and Knowledge Acquisition approaches currently being used, and planned to be used inside the Mobis platform. Related with task 4.2 “Knowledge Capture Services for Mobility Network”, this document provides an overall description and also some examples, of the methods developed for knowledge capture from documents and low-cognitive-load and natural language elicitation.

The main objective here, is to support reasoning with new knowledge, coming from external data sources, which describe the actual traffic network topology, transportation means and paths, transportation types. The new knowledge acquired and semantically represented, enables a traffic network object to better understand its real situation and also the souring environment.

Some of the methods presented here, represent an innovative development in the area of knowledge acquisition and representation done in the scope of Mobis project, while some of the methods extend, or rely on the previous work in the field of the knowledge acquisition plug-ins for the MobiS KB supported by Cyc reasoning engine. This deliverable builds upon the methodological approach to be followed under WP4 (detailed in D4.1 “MobiS knowledge base”), and also describes in detail the modules related with knowledge acquisition and representation, illustrated in D2.2 “MobiS technical requirements and system architecture”.

All of the approaches described here have in common that the knowledge they manage to capture uses the Mobis vocabulary and can be stored in the Mobis knowledge base, to be then used by the other parts of the system. The approaches here fit nicely as well with the overall Mobis architecture, trying to cover knowledge capture in the field of crowdsourcing (mobile app), call and traffic centers and the news sources.

2 Introduction

2.1 Terms and conventions used in the document

The following acronyms and abbreviations are used in the present document:

TERM	DEFINITION
CycKB	Cyc Knowledge Base
CycL	Predicate Logical Language used in CycKB
CURE	Content Understanding, Review, or Entry
NL	Natural Language
KA	Knowledge Acquisition
NER	Name-Entity Recognition

2.2 Structure of the document

Taking into account the above mentioned objectives and scope of this document, the deliverable at hand is broken down into the following main sections:

- Chapter 2 – Introduction: This section gives a brief structure overview of the document and explains the terms and conventions used in the document.
- Chapter 3 – Dialog Knowledge Acquisition: This chapter presents a novel approach to Knowledge Acquisition using natural language dialog with the users. First section explains how it is implemented. Second sub-section presents the crowd sourcing specifics to the implementation and the third section presents adapted implementation of the system in a traffic call center setting.
- Chapter 4 – Knowledge Capture from Documents and Low-Cognitive Load or Unstructured Data: This chapters is separated into two sections. First one describes the enrichment of NL texts with the structured data, where the second section describes a pattern approach to parse NL texts and convert them into the logical statements which could be put into the Mobis KB. The patterns can be made manually or semi-automatically.
- Chapter 5 – Conclusions: This section presents the conclusions and key findings of this deliverable.
- Chapter 6 – Annex: In this section we attached the relevant papers and knowledge extension documents done in the scope of the Mobis project and are relevant to this deliverable.

3 Dialog Knowledge Acquisition

Among the relevant components in Mobis platform are the Knowledge Acquisition (D2.2 Chapter 6.4.4) and Natural Language components (D2.2 Chapter 6.4.3) which, combined with an ontology (CycKB) and reasoning engine, can be used to elicit knowledge from the users in a natural language. This means that the user is presented with:

a) A natural language form which can be filled-in,
b) A Natural language question, which can be answered in a free text,
c) A natural language question or form, with pre-defined options which can be used to answer the question or fill-in the form,
d) A combination of a, b and c, where a free text can be used, or a pre-defined choice.

This functionality is based on the combination of preexisting Cyc technology called CURE [1] (Content Understanding, Review, or Entry) which generates patterns, and an independent approach, parts of which are being developed inside Mobis.

The main idea of this approach is to close the gap between the natural language user feedbacks and responses which computers can't understand, and the pure form like knowledge acquisition which can only elicit one very simple type of data (item selections, numbers, etc...). With this approach users are to some extent able to respond and interact in the natural language, and the computer is able to understand the responses and store them in a structured way into the knowledge base.

3.1 Implementation

The operation of the Knowledge Acquisition module relies heavily on the Mobis Knowledge base (D4.1) and its knowledge representation logical language which itself is based on CycL [2], the Inference Engine and Logic to NL generation capabilities of Cyc [3].

Typically knowledge capture process with inference support consists of the following steps, which all depends on the specific logical vocabulary:

1. Deciding when and what to ask,
2. Converting a logical question into a natural language,
3. Checking the consistency and existence of the answer,
4. Asserting the answer into the ontology and/or creating new concepts supporting it.

Each step depends on a logical vocabulary inside the KB that is used to describe the rules and assertions that control the inference engine and natural language responses. These steps are explained in a detail in the sub-chapters that follows on the example when user enters more info about the traffic incident and the place where he eats lunch[1].

3.1.1 Definition of the vocabulary

For the reasoning engine, to be able to decide when and what to ask, we need a special vocabulary which can store user context and define the formulation of the questions. This vocabulary depends on the pre-existing vocabulary of the Mobis Knowledge base (Cyc Upper ontology[2]).

The example of the logic vocabulary that defines two predicates[3] is presented in the following table:

[1] The lunch example is here for 2 reasons. Because the KA system was originally tested on the places people often visit due to the easiest testing and because this kind of knowledge is very useful for the later task of routing based on the personalized user knowledge.

[2] The upper ontology is the vocabulary covering basic definitions of the world e.g. Abstract and Concrete things, and the abstract descriptions of the relationships of those things to, for example, events, accidents, object, traffic; this is very important for the inference engine, but is not generally usable by, and presented to the users.

[3] A predicate is a relation between logical terms; for example the predicate *reportedEvent* is used to express a relation between particular event instance and the user who reported it: *(reportedEvent User1 TrafficAccident5)*.

```
Concept: lastVenue
(isa lastVenue BinaryPredicate)
(arg1Isa lastVenue User)
(arg2Isa lastVenue Venue)

Concept: reportedEvent
(isa reportedEvent BinaryPredicate)
(arg1Isa reportedEvent User)
(arg2Isa reportedEvent Event)
```

Table 1: Example of logical vocabulary defining user context (predicate for last user venue and the event user reported)

```
(lastVenue TheUser1 TheBar2)
```
(1)

The upper assertion is stating that last known location of the "User 1" was the "Bar 2". Another example is an assertion saying that the User 3 reported the "Event 5" (2).

```
(reportedEvent TheUser3 TheEvent5)
```
(2)

This can be then picked up by inference, to infer other things like possible activity for the user or type of event, or to propose an additional question to retrieve more data. For the inference engine to be able to do that, the vocabulary needs to support it:

```
Concept:  typeOfPlacehttp://localhost:3663/a/ftvaz/cg?cb-cf&c279723
(isa typeOfPlace BinaryPredicate)
(arg1Isa typeOfPlace Place)
(arg2Genl typeOfPlace Place)

Concept:  typeOfEventhttp://localhost:3663/a/ftvaz/cg?cb-cf&c279723
(isa typeOfEvent BinaryPredicate)
(arg1Isa typeOfEvent Event)
(arg2Genl typeOfEvent Event)

Concept: secondaryTypeOfPlacehttp://localhost:3663/a/ftvaz/cg?cb-cf&c279723
(isa secondaryTypeOfPlace TernaryPredicate)
(arg1Isa  secondaryTypeOfPlace Place)
(arg2Genl  secondaryTypeOfPlace Place)
(arg3Genl  secondaryTypeOfPlace Place)
```

Table 2: Example of vocabulary allowing more detailed knowledge about the places and events

The predicates above can be used for storing additional knowledge about the events and places (simple type in this case). For example an assertion telling us that that the "Bar 2" is a bar:

```
(typeOfPlace TheBar2 Bar-Organization)
```
(3)

And assertion stating that the "Event 5" is a traffic accident:

```
(typeOfEvent TheEvent5 VehicleAccident)
```
(4)

The second predicate *secondaryTypeOfPlace* allows us to state additional type information (e.g. that, in addition to being a bar, Bar 2 is a Cafe as well). This predicate is, from a logical point of view, redundant because it is possible to have multiple typeOfPlace assertions with the same first argument and different second arguments representing multiple types assigned to and further describing the same place. However, as explained above, this second predicate supports an interaction design that is

used later when communicating with the user (described in detail in next chapter). The second predicate, allows the system to talk about two types of places at once. The following assertion allows the system to say "Besides a kind of bar, The Bar 2 is also a cafe".

```
(secondaryTypeOfPlace TheBar2 Bar-Organization Cafe-Organization)
```
(5)

Even more helpfully, when eliciting knowledge, as in:

```
(secondaryTypeOfPlace TheBar2 Bar-Organization ?WHAT)
```
(6)

Because in the upper assertion (6) the ?WHAT is unknown, it enables asking: "Besides a kind of bar, what other kind of place is The Bar 2". Some detail on how this NL is generated is given in the following chapter (3.1.3).

The last and the most important part and of the vocabulary needed for the knowledge acquisition is used by the reasoning system to make it clear to the NL part of the application that it intends to ask the questions about something:

```
Concept:  kaWantsToAskUser
(isa kaWantsToAskUser TernaryPredicate)
(arg1Isa kaWantsToAskUser User)
(arg2QuotedIsa kaWantsToAskUser CycLOpenExpression)
(arg3Isa kaWantsToAskUser List)
```

Table 3: Definition of the predicate that triggers the process of asking the user a question

This predicate, when used in a logical statement, includes functional information about the question. Which user to ask, what to ask and the possible suggested answers which can be predefined or inferred from the user context. Below is an example of such assertion.

```
(kaWantsToAskUser TheUser1
  (secondaryTypeOfPlace TheBar5 Bar-Organization ?ANSWER)
  ((TheList
     Restaurant-Organization
     Cafe-Organization
     Nightclub-Organization))
```
(7)

This logical assertion means that the system intends that User 1 is asked what, besides the bar, The Bar 5 might be. The list "restaurant, café, nightclub" servers both to clarify what is being asked, and to provide a quick means of answering the most common cases. Another example of such assertion is the following.

```
(kaWantsToAskUser TheUser3
  (typeOfEvent TheEvent5 ?ANSWER)
  ((TheList
     VehicleAccident
     TrafficJam))
```
(8)

This assertions tells the system to ask User 3 what type event he reported, with suggestions for traffic jam or accident. See (2).

3.1.2 Rules that drive the knowledge acquisition

In previous paragraph we defined the vocabulary that supports the knowledge to be asserted into the KB and the inference to be able to decide when to ask specific questions. Here we define the forward inference rules which actually do the work:

```
Rule:
(implies
   (and
      (suggestionsForKaQuestionType
         VenueTypeOfPlace-KaQuestion ?SUGGESTIONLIST)
      (lastVenue ?USER ?VENUE))
kaWantsToAskUser ?USER (typeOfPlace ?VENUE ?SUGGESTIONLIST))
Meta Requirement:
(unknownSentence(thereExists ?PLACETYPE   (typeOfPlace ?VENUE ?PLACETYPE)))
```

Table 4: Logical rule that triggers the question to be asked

This rule is used by forward inference to trigger the KA system's "wish" to ask the question about the last place the user was. The rule is basically saying that for all users, if the type of their last venue is unknown, the system should ask about the type of the venue with a suggestion list that is defined for this type of question. This suggestion list is defined with other rules or direct assertions, but the details are beyond the scope of this explanation.[4]

In order to get more specific knowledge about the place, which follows after at least something is known, we need another rule which takes into account the previous answer and produces more detailed question:

```
Rule:
(implies (and
   (isa ?VENUE Bar-Organization)
   (lastVenue ?USER ?VENUE)
 (suggestionsForKaQuestionType SecondaryTypeOfPlace-Question
?SUGGESTIONLIST))
(kaWantsToAskUser ?USER
 (secondaryTypeOfPlace ?VENUE Bar-Organization ?TYPE) ?SUGGESTIONLIST))
```

Table 5: Follow-up Logical Rule that triggers additional question

This specific example rule only triggers when someone who is at the venue answers that this venue is a bar. Then the system asks what else the particular venue is. The results of this rule are an assertions like the example assertion (7). After it will be converted into English this assertion will be: "Besides a kind of bar, what other kind of place is The Bar 5".

Note that this rules are provided just as an example which we use to explain the functionality and possibilities of the system. The KA system in Mobis case has rules connected to traffic events and situations and the user itself, to be able to use this knowledge in later reasoning about the routes. Some of these rules are used in the knowledge acquisition implementation used in traffic centers which is explained in the chapter **Error! Reference source not found.** (KA in traffic centers).

3.1.3 Natural Language Generation

Inferring logical statements (assertions) that represent conversational intentions is only one of the steps in the KA process. These statements then has to be converted into natural language which is then being shown to the users. For this we use a Logic to NL conversion of Cyc system, which depends on

[4] Whether that wish is acted on depends on the state of the user interface and other contextual information. It is also worth noting that this rule is simple for expository purposes – The KA system desires to ask this question whenever a user is in a place *and* it does not already have an answer. The KA system gets this information from the mobile application (Mobis Commuting Assistant)

the linguistic assertions inside the KB[5]. The example of such assertions is presented in the following table:

As a question	As a statement
(genTemplate-QuerySentence (secondaryTypeOfPlace :PLACE :PRIMARY ?SECONDARY) (ConcatenatePhrasesFn (ConcatenatePhrasesFn (PhraseFromStringFn "besides a kind") (ConcatenatePhrasesFn-NoSpaces (Pp-PNpFn Of-TheWord (ParaphraseFn-Constrained nonPlural-Generic :PRIMARY)) (PhraseFromStringFn ", what other kind of place is"))) (ParaphraseFn :PLACE)))	(genTemplate secondaryTypeOfPlace (ConcatenatePhrasesFn (ConcatenatePhrasesFn (PhraseFromStringFn "besides a kind of") (ConcatenatePhrasesFn-NoSpaces (Pp-PNpFn Of-TheWord (ParaphraseFn-Constrained nonPlural-Generic :ARG2)) (PhraseFromStringFn ","))) (PhraseFn-Tensed :ARG1 (Sentence-NpIsXpFn (ParaphraseFn :ARG1) (ConcatenatePhrasesFn (PhraseFromStringFn "also") (Np-DetNbarFn-Indefinite (PhraseFn-Constrained nonPlural-Generic (ParaphraseFn :ARG3))))))))
Input: (secondaryTypeOfPlace TheBar5 Bar-Organization ?WHAT)	Input: (secondaryTypeOfPlace TheBar5 Bar-Organization Café-Organization)
Result: “Besides a kind of bar, what other kind of place is the Bar 5?”	Result: “Besides a kind of bar, Bar 5 is also a cafe.”

Table 6: Example of the declarative and interrogative NL assertions

3.1.4 Consistency check and asserting the new knowledge

When the user answers on the proposed question, the KA system checks whether the answer is something it knows already or it is completely new concept. It checks as well if the answer is actually consistent with the pre-existing knowledge. This is done with the help of the vocabulary definitions, especially with the type constraints on predicate arguments. Other consistency checks, including attempting to prove the opposite of the proposed assertion, may also be used.

The first thing the system does after retrieving the answer from user is trying to find appropriate concepts based on the NL string provided by the user. It does that by searching over the possible matches which satisfies the ontology vocabulary for the specific answer. If we use the example where it is asking about the type of the place (8) the query for the inference engine looks like this:

```
Query:
(#$and
  (#$termStrings ?TERM "traffic accident")
  (#$or
```

[5] Similarly as any other logical assertions, for example assertions describing what and when to ask, there is a vocabulary which can be used to tell the inference engine how to generate language out of the logic.

```
    (#$and
      (#$unknownSentence
        (#$thereExists ?ARGENL (#$argGenl typeOfEvent 2 ?ARGENL)))
      (#$equalSymbols ?ARGENL #$Nothing))
    (#$and
      (#$argGenl typeOfEvent 2 ?ARGENL)
      (#$genls ?TERM ?ARGENL)))
  (#$argIsa typeOfEvent 2 ?ARGISA)
  (#$isa ?TERM ?ARGISA))
```

Table 7: Query that checks for type consistency and finds the concept that matches answer string

This query returns the concept for the traffic accident (#$VehicleAccident) which can be used as an answer without more disambiguation since it is the only result and fits the ontology constraints. The final result of this is the assertion (4).

In cases when the query returns more than one answer (because there are many possible types with the same name, due to polysemy), the user is presented with options to pick one among the possible answers or to define a new one – the NLG system attempts to describe the choices as differentially as possible.

If the above query returns no results, then the KA system searches the full KB to find an appropriate type based on its English name string. If it finds one then it uses inference to check whether that answer is provably disjoint with the constraints or not. If there is no result with that as well, then it creates a completely new concept which fits an ontology constraints. This way a new types of traffic events, places, etc. can be created.

3.2 Crowdsourcing

The idea of the KA approach described in the chapters above is that it is integrated into the Mobis Commuting Assistant application (Figure 1) and used on a larger scale of users.

Figure 1: Prototype of Mobis Commuting Assistant Application

This users are then, among their location and speed of travelling contributing a various knowledge to the system. This can sometimes be contradictory because of different opinions, or it just simply changes by time. Because Mobis has one general Knowledge Base, there needs to be some way to handle and discover these inconsistencies.

Inside Knowledge Base, the logical assertions can be asserted into a different places or Microtheories (similarly like different named graphs in RDF). These Microtheories can be hierarchically connected, where more specific Microtheories can access the knowledge in more general ones, but not in the other direction. This way it is possible to direct the inference engine, where to look for the knowledge and "override" the specific assertions. For clearer understanding please refer to Figure 2, where it is clear that what is asserted in the *User2Mt* cannot be seen in *User1Mt*, whereas both Microtheories can accesses the assertions from *TrafficMt* and *UniversalMt* respectively.

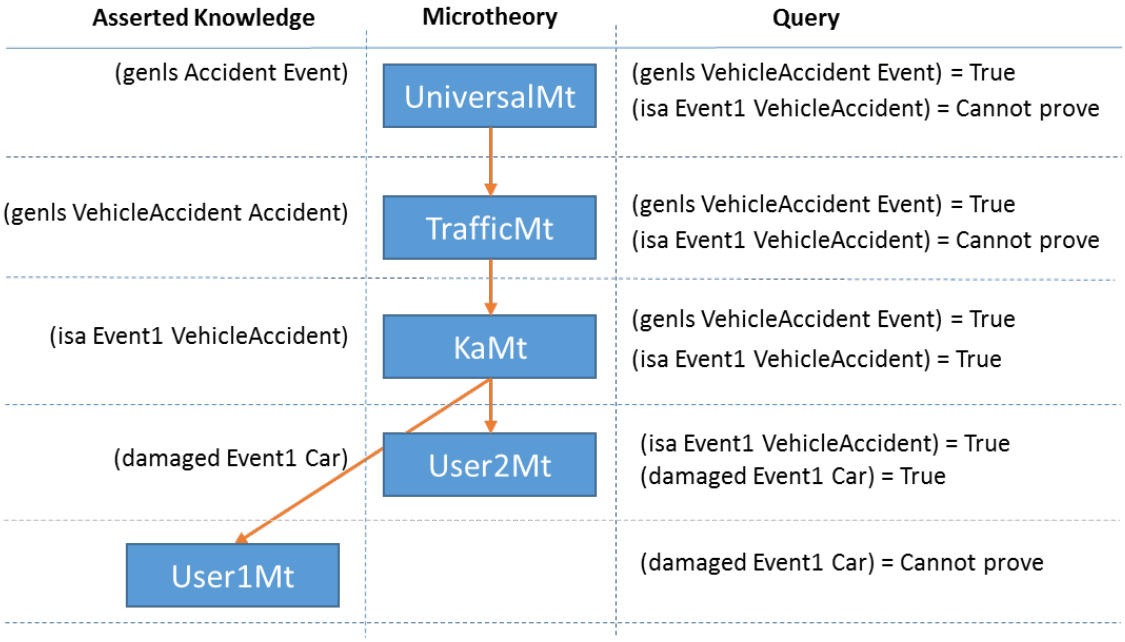

Figure 2: Knowledge Base Microtheory structure example

Inside the Knowledge Acquisition module we use these Microtheories to solve different opinions of users and the changes through time. There are two approaches, which need to be tested through the proceeding of the Mobis project.

The first approach which and currently the default one is that the acquired knowledge (except the personal data), goes directly into the general *KaMt*. Each assertion from there is then presented to the other users as a checking question. For example: "Is it true that this is a traffic accident?", "Is it true that The Bar 5 is a type of Bar". The user then has to confirm or negate the statement. The system then counts the number of agrees and disagrees and if the disagreement comes over some threshold, the assertion is removed.

The second approach goes into another direction. Each answer is always answered to the User Microtheory (*User1Mt*, *User2Mt* …). The KA system then counts how many users gave the same answer to the same question. When this comes over some threshold, the assertion is moved into the more general *KaMt*, where it can be seen by other users of the system as well.

3.3 KA in traffic call centres

In the scope of project KA objectives, a system for traffic call centers is being developed. Its main goal is to help (often inexperienced) call operators to effectively obtain relevant information about the traffic (accident, congestion, broken car, etc…) report call when it happens. It enables the users to quickly enter the more exact knowledge about the traffic situation or event (like position of the car on the road in the case of an accident, or a length of the queue in the case of congestion). This information is then stored in the KB and accessible to other applications (and users) in the Mobis platform.

Similarly as the approach described in chapter 3.1, the system works on top of the Mobis Knowledge base (which itself is based on CycKB - D4.1). It depends as well on the NL Module which uses inference rules and NL knowledge inside the KB to form the questions for the user. The NL module share some functionality with the Dialog KA (3.1). The overall functionality and how it fits into broader Mobis system is described in the D2.2 (Technical Requirements and System Architecture).

User communicates with the system over the list of natural language questions and responses. The response is then parsed and asserted into the KB in a similar way as described in the chapter 3.1. To achieve that, the system uses parts of the same components as well. The lexicon, which is the main component of the NL system, contains information about English words. Each word, or a combination of words is represented as some logical concept. The system also provides NL generation from its internal logic language knowledge representation. This is achieved by templates that are specific enough to represent the mapping from logic precisely. They use predicates and functions that describe linguistic characteristics of a certain word or sentence.

Within Mobis, the initial version of this KA support system is planned to be installed inside the AMZS call center, where it should support the call operators to effectively gather the relevant knowledge about the problems which were behind the call. This particular implementation of the KA application can be seen as an Expert System (ES), with an inference engine and knowledge-base that uses KB driven NL dialogs as a communication basis between domain experts, knowledge engineers and call operators. The ES architecture is represented in the Figure 3.

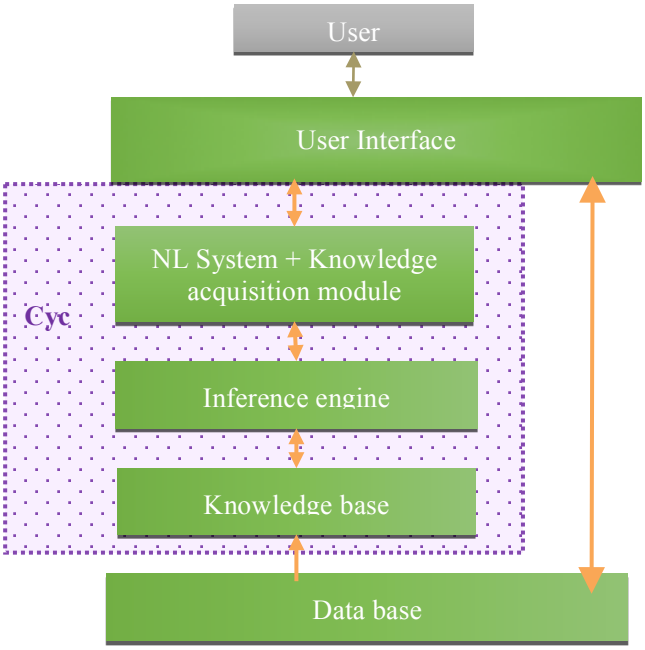

Figure 3: Knowledge acquisition support system architecture

The system guides the operator through a sequence of questions that are chosen individually in each issue corresponding to the topic of the event. Questions are chosen and generated by Knowledge base according to the rules. There is also an additional reasoning module that is able to identify a car malfunction based on the previously collected data, or at least suggest an appropriate solution.

When the question is posed, the user's answer can be a fact that is already a known concept in the Knowledge base. If not, new concept is created. When the question is answered that information is saved in Mobis knowledge base. All of the newly acquired data is available for queries and rules and other manipulation in the context of Mobis. The example of that can be seen in the Figure 4, showing the KA about a vehicle accident. Questions about it are divided into different tabs according to the theme they're about. In the "About the accident" tab the system asked about the exact location, which is the road (it could also be a ditch, a cliff etc.) and the vehicle orientation, which is in this case upside-down. The assertions about the car being stationed in the road then triggers rules about that certain traffic event in the knowledge base. Thus, warnings about traffic jams can be released automatically. Also, based on accident's location and answers to questions in "Technical information" tab an estimate about the duration of clearing the road can be given.

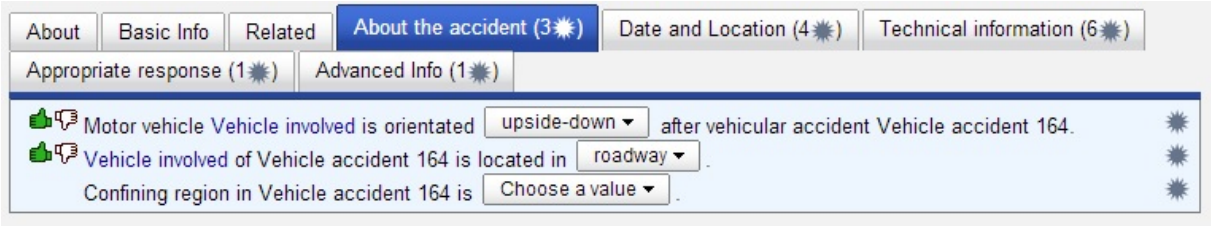

Figure 4. Call centre Knowledge acquisition example.

For the future work, besides the main functionality described above, the system could be easily extended by additional rules that could be used to reason about what would be the appropriate response for a reported event. For example it could be used to decide which vehicle is the most appropriate to be sent for help to consume as little gas as possible by not sending towing truck when not needed.

4 Knowledge Capture from Documents and Low-Cognitive Load or Unstructured Data

4.1 Knowledge Capture through Semantic Enrichment, Knowledge Source Classification and Name-Entity Recognition

The first prototype for knowledge capture on documents and other types of unstructured data is based on three distinct sub-processes: Semantic Enrichment, Knowledge Source Classification and Name-Entity Recognition. The first will create a vector of semantic concepts which helps to describe the knowledge source in terms of Cyc concepts, in order to assert new knowledge into the Cyc Knowledge Base about MobiS-relevant events (accidents road blocks, etc.). The semantic vector will also be used on the Knowledge Source Classification process, which is simply a clustering algorithm that classifies the input knowledge sources, pruning the ones that are not relevant for MobiS context. The third and last process links the asserted knowledge done by the Semantic Enrichment process with location-based information, gathered through a Name-Entity Recognition (NER) procedure. This process uses NER and Geocoding algorithms to find relevant, location-based information to add to the Cyc KB.

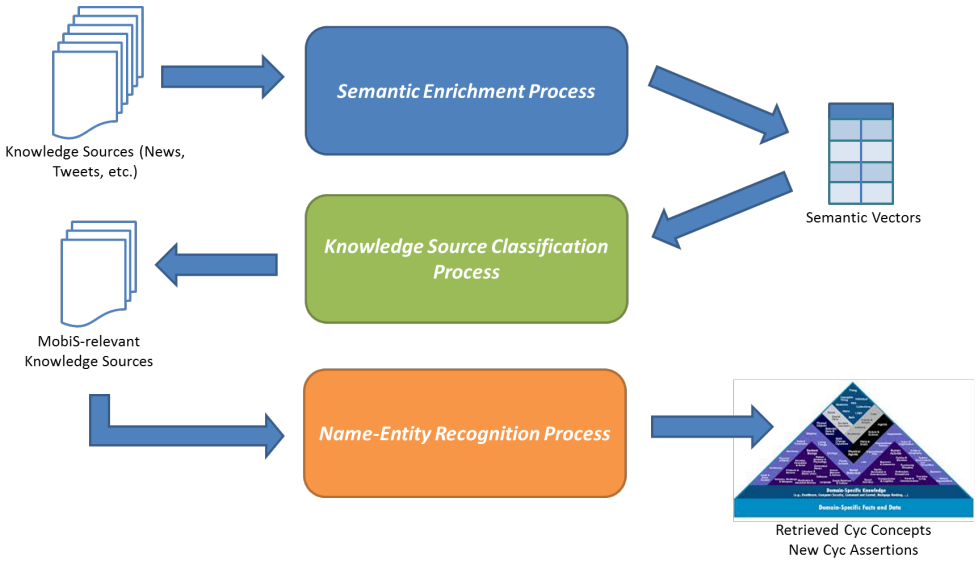

Figure 5. Knowledge Capture process through Semantic Enrichment, Knowledge Source Classification and Name-Entity Recognition

4.1.1 Knowledge Sources

MobiS uses several kinds of low-cognitive load or unstructured knowledge sources. These sources are mainly composed by free text, which is gathered from news feeds, tweets, documents or other kinds of news sources. For the initial tests, several sources were chosen, such as the IJS Newsfeed [4] (http://newsfeed.ijs.si), the Greek Twitter page @MyRouteGr, which specializes in traffic news and the Swedish DATEX II API [5], which also has a service for real-time traffic info and events, as well as other news Web pages and portals.

The IJS Newsfeed is accessed through an API, which returns the latest xml-formatted batch of news, segmented by time into .gz files, each a few MB in size. The stream is accessible at http://newsfeed.ijs.si/stream/. The URL accepts an optional `?after=TIMESTAMP` parameter, where TIMESTAMP takes the ISO format `yyyy-mm-ddThh:mm:ssZ` (Z denotes GMT time zone). The server will return the oldest gzip created later than TIMESTAMP.

In order to access the @MyRouteGr and other traffic related tweets, a Twitter REST API [6] interface was implemented in Java. The Twitter REST API provides a Twitter Search service, which returns a collection of relevant Tweets matching a specified query.

The information supplied by the Swedish Transport Administration is sent as messages in Datex II format, which is based on an XML schema. Datex II uses web service technology to deliver traffic information. The use of web services is recommended because it provides greater flexibility across platforms and a possibility of better error handling. The specific DATEX II traffic message service from Trafikverket is provided via the URL:

https://datex.trafikverket.se/d2clientpull/situationpullserverBA/2_0/TrafficMessageService/TrafficMessageService.asmx

4.1.2 Semantic Enrichment Process

The Semantic Enrichment process is comprised by two different modules: The Document Analysis Module has the knowledge extraction mechanisms that process raw information from knowledge sources (e.g. text, metadata) and extract the most relevant terms from that information, in terms of their occurrence frequency and position in such sources, filtering unwanted or irrelevant terms which occur many times on a source, but do not introduce any relevance to its knowledge representation; The Semantic Enrichment Module handles knowledge source indexation, which manages the creation of knowledge sources' semantic vectors, constructing a knowledge representation not only of the source itself, but also of the underlying meaning inherent to its most relevant terms.

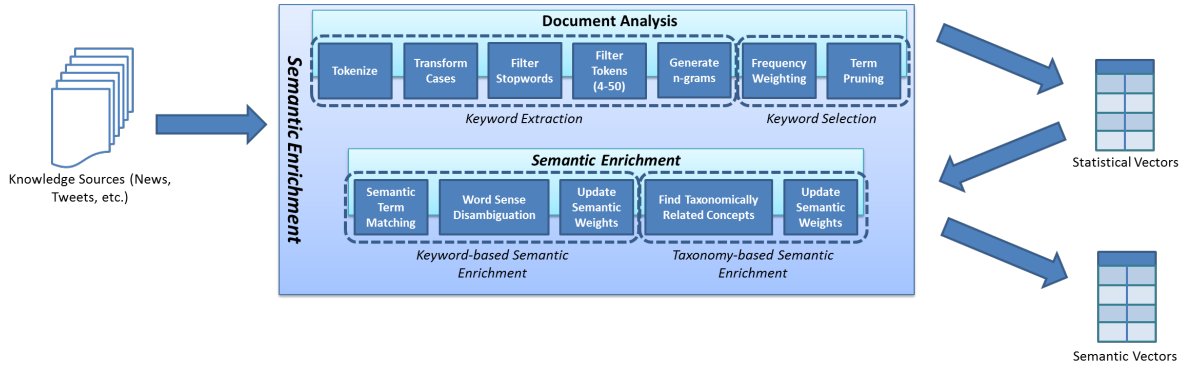

Figure 6. Semantic Enrichment Process

4.1.2.1 Document Analysis Module

Knowledge extraction is usually a process embracing three stages: word extraction, regular expressions filtering, and statistic vector creation. Word extraction is the process in which words and expressions are extracted from a knowledge source and divided by data- and text-mining techniques. Text mining corresponds to the extension of more traditional data mining approach to unstructured textual data and is concerned with various tasks such as extraction of information implicitly contained in collections of knowledge sources, similarity-based structuring and visualization of large sets of texts [7]. Another responsibility of text mining tools is regular expression filtering, which is the process of removing frequently occurring terms that have no relevance for the context of a particular source, but normally appear several times in all knowledge sources, such as grammatical articles (the, a/an, some), pronouns (mine, yours, me, she anybody, etc.), prepositions (for, in, under, toward, etc.) or adverbs (much, few, quite, slowly, etc.).

This process is performed by noise reduction and filtering procedures, such as stop word filters. The last stage, statistic vector creation, is the process that builds the statistical representation of a knowledge source, through the analysis of each term extracted from the source, in terms of its frequency, emphasis and position within the corpus of such source. The statistic vector is organized in matrix form and is composed by the extracted terms, or keywords, and by the statistical weight of each keyword within the knowledge source, based on the frequency analysis introduced above. This vector of keywords and respective weights is represented by an approximation to the term-document matrix vector, which is denominated statistic vector.

There are many state-of-the-art data-mining and text-mining frameworks and techniques available. Knowledge extraction techniques are the basis for IR, supporting syntactical comparison between users' queries and words or expressions contained in knowledge sources. Most search of today's Web search engines function according to this principle. The software used to create statistic vectors was RapidMiner [8], which is a knowledge extraction tool in the form of an easy-to-use IDE, and that builds statistic vectors of knowledge sources over sources' corpora. RapidMiner has several filter tools, such as stemming and lemmatization filters. The whole extraction process is as follows:

a) First of all, each document is broken into sentences. Then, terms in each sentence are extracted as tokens (this process is called tokenization).
b) All tokens found in the document are transformed to lower case.
c) The terms belonging to a predefined stop word list are removed.
d) Tokens whose length is "< 4" or "> 50" characters are discarded.
e) The n-Grams generation is seen here as a creation of sequences of 1 to N words. For this case we are considering the generation of unigrams, bigrams (e.g. waste management) and trigrams (e.g. electric power product).

Terms of low frequencies are supposed as noise and useless, thus we apply the *tf–idf* (term frequency - inverse document frequency) [9] method to choose the key terms for the document set. Equation 1, is used for the measurement of $tfidf_{ij}$ for the importance of a term t_j within a document d_i. The main limitation of *tf-idf* method is that long documents tend to have higher weights than short ones. It considers only the weighted frequency of the terms in a document, but neglects the length of the document. In Equation 2, tf_{ij} is the frequency of t_i in d_j, and the total number of occurrences in d_j is the maximum frequency of all terms in d_j used for normalization to prevent bias for long documents.

$$tfidf_{ij} = tf_{ij} * idf_i \quad (1)$$

$$tf_{ij} = \frac{number\ of\ occurrences\ of\ t_i\ in\ d_j}{total\ number\ of\ occurences\ in\ d_j} \quad (2)$$

$$idf_i = \log \frac{number\ of\ documents\ in\ D}{number\ of\ documents\ in\ D\ that\ contain\ t_i} \quad (3)$$

After calculating the weight of each term in each document, those which satisfy the pre-specified minimum *tf–idf* threshold γ are retained. For this work, we consider all terms where its *tf-idf* score was greater or equal than *0.001*. Subsequently, these retained terms form a set of key terms for the document set *D*. A document, denoted d_i is a logical unit of text, characterised by a set of key terms t_j together with their corresponding frequency f_{ij}, and can be represented by:

$$d_i = \{(t_1, f_{i1}), (t_2, f_{i2}), \dots, (t_j, f_{ij}), \dots, (t_m, f_{im})\} \quad (4)$$

Such representation is entitled statistical vector, meaning that, for each document in *D* there is a resultant statistical vector. The statistic vector is organized in matrix form and is composed by the extracted terms, or keywords, and by the statistical weight of each keyword within the knowledge source, based on the frequency analysis introduced above.

4.1.2.2 Semantic Enrichment Module

The Semantic Enrichment Module performs knowledge source indexation tasks every day, at a scheduled time, for all sources uploaded during that day. The knowledge source indexation process's function is to create a semantically enriched knowledge representation of the knowledge source from the syntax-based statistic vector outputted by the Document Analysis Module [10]. This representation comes in the form of semantic vectors, which is a particular type of vector extracted from a document-term matrix, used in the knowledge extraction process. The difference between the general term-

document form and the more specific semantic form is that the semantic vector is composed by concepts which may not be directly present on the source, but have an inherent relation with the terms present within the statistic vector.

The terms and expressions present on the statistic vector are called keywords. Each keyword will be matched against the Cyc Knowledge Base (KB) [11], in order to extract concepts that are represented or expressed by the keyword, in terms of Natural Language. For instance, the keyword "*car*" will match with the Cyc concept *"#$Automobile*". This matching is done via a query to the Cyc KB, in the form:

- Query: *(#$termStrings ?CONCEPT "keyword")*

The *#$termStrings* predicate is one of many predicates used by Cyc KB to relate a concept to a set of terms that represent or express that concept in free text, such as the above example "*car*". Other predicate that may be used to the same end is *#$prettyString*. The query returns a list of concepts which are expressed by the keyword or an empty list, if no concept is returned in the query results.

Cyc already provides a concept tagger web service, as part of its tools. The tagger receives as input a portion of text, such as a keyword or a set of keywords, and returns the keywords with the matched Cyc concepts. The text may be tagged against the whole Cyc KB or only a specific microtheory. The tagger has also a Name Entity Recognizer tool, which is used to recognize know entities, such as people names or addresses.

Hence, all keywords on the statistic vector are matched against the KB via the concept tagger service. The next step is to disambiguate retrieved concepts, because one keyword may have more than one concept associated to it. In this case, the idea is to know what microtheory has more retrieved concepts associated to it. For instance, if the knowledge source is some news about traffic, it is plausible that most concepts will be in microtheories related to transportation, as vehicle and road names and types. Therefore, the disambiguation between retrieved concepts is done in several steps:

- Get the corresponding microtheory for each retrieved concept, via the query
  (#$definingMt #$Concept ?MT)
  The *#$definingMt* predicate associates a concept to a particular microtheory;
- Check what microtheories have more retrieved concepts associated to them;
- When one keyword has more than one concept associated, check if any of these is present in the predominant microtheories;
- If so, remove the other associated concepts from the concept vector;
- If not, check if there is some semantic relation between any retrieved concept associated to the predominant microtheories and other concepts that are not related to the predominant microtheories. For this, we use the query
  (?PRED #$PredominantMtConcept #$OtherConcept)

If the query returns any result, it means that there is a concept that is not associated with the predominant microtheories but is semantically related with those microtheories. For instance, the concept *#$CarAccident* is not in the *#$TransportationVocabularyMt* microtheory, but it is related with the concept *#$Automobile*, which is associated with this microtheory. In this case, car accident would be added to the concept vector;

If the above two steps do not manage to disambiguate concepts related to a particular keyword, the last step is to use the Taxonomy-based semantic vector creation process, described below;

4.1.2.2.1 Semantic Vector Creation

Semantic vector creation is the basis for the presented approach, as it represents the extraction of knowledge and meaning from documents and the agglomeration of this information in a matrix form, better suited for mathematical applications than the raw text form of documents.

A semantic vector is represented as a matrix with two columns: The first column contains the concepts that build up the knowledge representation of the document, i.e. the most relevant concepts for contextualizing the information within the document; the second column keeps the degree of relevance, or weight, that each term has on the knowledge description of the document [12].

The presented approach takes into account two different, but complementary procedures for building up the semantic vector, each of which considered a more realistic iteration of the knowledge representation of a document: Keyword-based and taxonomy-based semantic vectors.

The keyword-based semantic vector creation comprehends the matching of keywords from a statistic vector with keywords present in the Cyc KB, the subsequent extraction of the concepts related semantically with matched keywords, and the attribution of semantic weights to each concept, depending on its relevance within the knowledge sources' universe.

The first two functions, namely keyword matching and concept extraction, are based on extracting all the keywords and equivalent terms associated to concepts in the ontology, comparing them with the statistic vector's keywords, and subsequently creating a list of all the concepts on the KB which have one or more matched keywords associated. Such process depends on Knowledge Extraction Module results and can face some issues. One problem is that keywords are not formed only by one word. Both statistic vector and ontology have keywords with two or more words, but statistic vector's keywords contain only stemmed words. The presented work approaches this problem by only comparing statistic vector and ontology keywords that have the same word count. The statistical weight for each concept is given by:

$$w_c = \sum_{i=1}^{n} w_{k_i} \tag{5}$$

In Equation 5, n is the total number of keywords from the statistic vector that are associated with the concept, w_c is the resultant statistic weight for the concept and W_{k_i} is the statistic weight for keyword associated to the concept. This means that the total statistic weight of a concept within a knowledge source is defined by the sum of the statistic weights assigned to the keywords associated with the concept, if the source contains such keywords, or zero otherwise. For instance, if an ontological concept has two keywords associated to it, with statistic weights 0.05 and 0.02, respectively, then the total statistic weight for that ontological concept would be 0.07.

4.1.2.2.2 Keyword-based Semantic Vectors

The result of the application of Equation 5 on all ontology concepts extracted from a knowledge source is a vector comprising those concepts and the corresponding statistical weights. In this stage, the resultant vector is not yet considered a semantic vector.

The next step further into the knowledge source indexation process is the attribution of semantic weights to each of the concepts. A semantic weight is a weight attributed to a concept according to its semantic meaning within the construction domain, and regarding not only the current source's context, but also the different contexts of all knowledge sources comprised within the system. This approach uses an approximation to the TF*IDF family of weighting functions to calculate the semantic weight for each concept resultant from the concept extraction process. The TF*IDF approximation algorithm used is given by the expression:

$$w_x = \frac{w_{x,d}}{max_y w_{y,d}} \cdot \log \frac{D}{n_x} \tag{6}$$

In Equation 6, $w_{x,d}$ is the statistical weight for concept x in knowledge source d's statistical vector, $max_y w_{y,d}$ is the statistical weight of the most relevant concept, y, within the statistical vector of knowledge source d, D is the total number of knowledge sources present in the search space, n_x is the number of sources present in the search space which have concept x in their semantic vectors, and w_x is the resultant semantic weight of concept x for knowledge source d.

The keyword-based semantic vector is then stored in the database in the form $\left[\sum_{i=1}^{n} x_i , \sum_{i=1}^{n} w_{x_i}\right]$, where n is the number of concepts in the vector, x_i is the syntactical representation of the concept and w_{x_i} is the semantic weight corresponding to concept i. Statistical normalization is performed over the keyword-based semantic vector's weights, in order to obtain values between zero (0) and one (1). This normalization is computed as:

$$w_x^* = \frac{w_x}{\sum_{i=1}^{n} w_{x_i}} \tag{7}$$

In Equation 7, w_x is the semantic weight for concept x for knowledge source d, n is the total number of concepts in source d's semantic vector, w_{x_i} is the semantic weight for concept i, and w_x^* is the normalized semantic weight of concept x for source d. Hence, normalization is simply dividing each semantic weight by the total sum of semantic weights present in knowledge source d's semantic vector.

4.1.2.2.3 Taxonomy-based Semantic Vectors

The next iteration on semantic vector creation is to define a taxonomy-based semantic vector. Taxonomy-based vectors push one step further in the representation of KSs by adjusting the weights between expressions according to the taxonomic relation among them, i.e., expressions that are related with each other with the '*is_a*' type relation. If two or more concepts that are taxonomically related appear in a keyword-based vector, the existing relation can boost the relevance of the expressions within the KS representation. Furthermore, this process will disambiguate the remaining concepts that are related to the same keyword, by calculating the taxonomic distances between those concepts and the concepts belonging to the predominant Cyc microtheory, as referred earlier. In order to do this, this approach will take into account the notion of homologous and non-homologous concepts [13]:

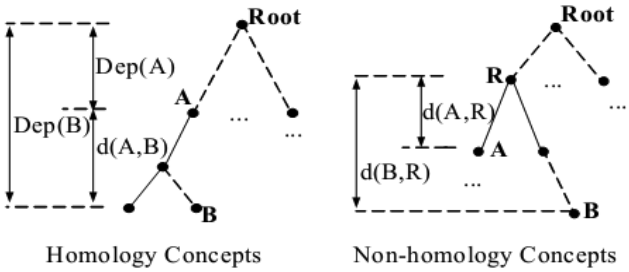

Figure 7. Homologous and Non-homologous Concepts

- Definition 1: In the hierarchical tree structure of the ontology, concept A and concept B are homologous concepts if the node of concept A is an ancestor node of concept B. Hence, A is considered the nearest root concept of B, R(A,B). The taxonomical distance between A and B is given by:

$$d(A,B) = |depth(B) - depth(A)| = |depth(A) - depth(B)| \quad (8)$$

  In Equation 8, depth (X) is the depth of node X in the hierarchical tree structure, with the ontological root concept's depth being zero (0).

- Definition 2: In the hierarchical tree structure of the ontology, concept A and concept B are non-homologous concepts if concept A is neither the ancestor node nor the descendant node of concept B, even though both concepts are related by kin; If R is the nearest ancestor of both A and B, then R is considered the nearest ancestor concept for both A and B concepts, R(A,B); The taxonomical distance between A and B is expressed as:

$$d(A,B) = d(R,A) + d(R,B) \quad (9)$$

The disambiguation method is to calculate the depth in relation to the root concept (in Cyc is the *#$Thing* concept) between the concepts from the predominant microtheory and the concepts to disambiguate. The one which is closer to the predominant microtheory concepts is maintained in the vector, whereas the furthest one is removed from the vector.

The taxonomy-based semantic vector is calculated using the keyword-based vector as input, where taxonomical relations are used to boost the relevance of the concepts already present within the vector or to add new concepts. The weight of the concepts is boosted when two concepts found in the keyword-based vector are highly relevant, with the degree of relevance being defined by a given threshold. If the relevance of the taxonomical relation between two concepts is higher than the

predefined threshold, then the semantic weight of such concepts is boosted in the taxonomy-based vector. If a concept already present in the keyword-based vector is taxonomically related to a concept than is not present in the vector, then the related concept is added into the taxonomy-based vector. The taxonomical similarity is calculated differently for both homologous and non-homologous taxonomical relations defined previously:

$$Sim(A,B) = \left(1 - \frac{\alpha}{depth(A)+1}\right)\frac{\beta}{d(A,B)}\frac{son(B)}{son(A)} \quad (10)$$

If $d(A,B) \neq 0$ and A and B are homologous.

$$Sim(A,B) = \left(1 - \frac{\alpha}{depth(R)+1}\right)\frac{\beta}{d(A,B)}\frac{son(A)+son(B)}{son(R)} \quad (11)$$

If $d(A,B) \neq 0$ and A and B are non-homologous.

$$Sim(A,B) = 1 \quad (12)$$

If $d(A,B) = 0$.

As previously referred, if two or more concepts are taxonomically related, this underlying relation may trigger two different processes [14]:

- ***Process 1: When C_x (an ontology concept that belongs to the semantic vector) is taxonomically related to C_y (another ontology concept), and C_y is also present on the semantic vector.***

In this case, the weights corresponding to C_x and C_y are boosted within the semantic vector, by applying the following equations:

$$tw_{C_x} = w_{C_x} + \sum(all\ related\ C_y's)\left[w_{C_y} \cdot \left(TI_{C_xC_y}\right) + Co - Occurrence_{C_xC_y}\right] \quad (13)$$

$$tw_{C_y} = w_{C_y} + \sum(all\ related\ C_x's)\left[w_{C_x} \cdot \left(TI_{C_xC_y}\right) + Co - Occurrence_{C_xC_y}\right] \quad (14)$$

Such weight boost is only performed if the taxonomic importance $TI_{C_xC_y}$ is greater or equal than a certain threshold. This constraint only accepts the weight boost if the two related concepts are linked by a relation that is strong (i.e. both concepts are taxonomically near in the ontology tree). $TI_{C_xC_y}$ is given by homologous and non-homologous similarity equations. In the case of Equation 12, the weight boost is not applied for obvious reasons. There are two different thresholds for, respectively, homologous and non-homologous concepts.

In Equations 13 and 14, tw_{C_x} and tw_{C_y} are the new taxonomy-based semantic weights for concepts C_x and C_y, respectively; w_{C_x} and w_{C_y} are the weights present on the keyword-based semantic vector, for concepts C_x and C_y, respectively. $Co - Occurrence_{C_xC_y}$ represents the frequency of occurrence of both concepts over the knowledge sources' search space, and it is computed by an approximation to the IDF formula:

$$Co - Occurrence_{C_xC_y} = idf\left(C_x, C_y\right) = \log\frac{D}{n_{C_xC_y}} \quad (15)$$

In Equation 15, D is the total number of knowledge sources in the system's repository, and is the number of knowledge sources that have concepts C_x and C_y in their keyword-based semantic vectors. Afterwards, the semantic vector's weights have to be normalized again, using Equation 7, so that each weight represents a percentage of relevance on the knowledge representation of a knowledge source again.

Parameters α and β, used in Equations 10 and 11, are adjusted in a different fashion, in order to reduce human intervention in the semantic vector creation process. Both parameters are calculated according to the usage of each kin relation within the knowledge source corpus' universe. More specifically in Equation 10, parameter β adjusts the importance of the distance between concepts, and parameter α adjusts the relevance given to the depth of root concept A. The existent homologous relation dictates that parameter β has a bigger relevance, because the depth of the root concept does not define the proximity relation between the homologous concepts. Hence, for Equations 10 and 11, parameters α and β are expressed as:

$$\beta = \frac{number\ of\ vector\ occurrences\ for\ taxonomically\ related\ pair\ A,B}{number\ of\ occurrences\ for\ the\ most\ frequent\ taxonomically\ related\ pair} \quad (16)$$

$$\alpha = 1 - \beta \quad (17)$$

Equation 16 states that parameter β is computed by dividing the total number of occurrences, in the system's search space, of the taxonomical pair (A,B), by the total number of occurrences of the most frequent taxonomical related pair of concepts in the search space. Parameter α is considered to be the inverse of parameter β, as stated by Equation 17, which implies that if concepts A and B appear often on the system's semantic vector universe, then the distance between them will be much more relevant than the depth of ancestor concept A. If, in contrast, the pair (A,B) is not a frequently occurring pair, then the relevance given to the distance between concepts is downgraded in favor of root concept A's depth.

- ***Process2: When C_x (an ontology concept that belongs to the semantic vector) is taxonomically related to C_y (another ontology concept), and C_y is not present on the semantic vector.***

In this case, C_x is not modified and C_y is added to the semantic vector, and its weight is computed as:

$$tw_{C_y} = w_{C_y} + \sum (all\ related\ C_x's) \left[w_{C_x} \cdot \left(TI_{C_x C_y} \right) \right] \quad (18)$$

$$tw_{C_x} = w_{C_x} \quad (19)$$

In this case, the system has to calculate the TF*IDF weight for concept C_y, w_{C_y}, which brings a conceptual problem: C_y does not possess any statistic weight resulting from the Knowledge Extraction process. The chosen approach in this case is to apply only the IDF term of Equation 1. This is made by attributing the value one (1) to the divisor and the dividend of the TF term of Equation 1, which means IDF is applied exactly according to Equation 3. As in the previous process, the new concept is only added to the taxonomy-based semantic vector if the taxonomic relevance, $TI_{C_x C_y}$, is greater or equal than a threshold. $TI_{C_x C_y}$ is computed as in the previous process.

4.1.3 Knowledge Source Classification Process

The main objective with the classification process, is to classify the knowledge sources identified previously into a pre-defined set of categories/clusters. The set of clusters are tightly related with traffic events which need to be identified. The initial set of clusters to be used for the classification process are described as: (i) queuing traffic, (ii) traffic accident, (iii) road closed, (iv) road works, (v) object on roadway and (vi) weather conditions.

We aim to use an unsupervised classification algorithm (K-Means clustering [17]) to classify the knowledge sources. The algorithm used for knowledge sources classification is described as follows.

Let a set of text documents be represented as a set of vectors X = {X1,X2,…,Xn}. Each vector Xj is characterized by a set of m terms (t1, t2,…, tm). m is the total number of unique terms in all documents which form the vocabulary of these documents. The terms are referred to as features. Let X be a set of documents that contain several categories. Each category of documents is characterized by a subset of terms in the vocabulary that corresponds to a subset of features in the vector space.

A simple illustration of text data in VSM is given in Table 8. Here, xj represents the jth document vector; ti represents the ith term; each cell in the table is the frequency that term ti occurs in xj. A zero cell means that the term does not appear in the related document. Documents x0, x1, x2 belong to one category C0, assuming "Climate Control", while x3, x4, x5 belong to another category C1, assuming "Waste Management".

Because these two categories are different, they are categorized by different subsets of terms. As shown in Table 8, category C0 is categorized by terms t0, t1, t2 and t4 while category C1 by terms t2, t3 and t4. In the meantime, terms play different roles on identifying categories or clusters. For instance, the same frequency of t4 appears in every document of category C0, hence, t4 should be more important than other terms in identifying category C0.

		t_0	t_1	t_2	t_3	t_4
C_0	x_0	1	2	3	0	2
	x_1	2	3	1	0	2
	x_2	3	1	2	0	2
C_1	x_3	0	0	1	3	2
	x_4	0	0	2	1	3
	x_5	0	0	3	2	1

Table 8: A Simple Illustration of Text Data in VSM

K-means algorithm finds a partition such that the squared error between the empirical mean of a cluster and the points in the cluster is minimized. Let μk be the mean of cluster Ck. The squared error between μk and the points in cluster Ck is defined as

$$J(C_k) = \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \quad (20)$$

$$J(C_k) = \arg\min_S \sum_{k=1}^{k} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \quad (21)$$

Minimizing this objective function is known to be an NP-hard problem (even for K = 2) [15]. Thus K-means, which is a greedy algorithm, can only converge to a local minimum, even though recent study has shown with a large probability K-means could converge to the global optimum when clusters are well separated [16]. K-means starts with an initial partition with K clusters and assign patterns to clusters so as to reduce the squared error. Since the squared error always decreases with an increase in the number of clusters K (with J(C) = 0 when K = n), it can be minimized only for a fixed number of clusters.

The reasons why unsupervised classification was chosen over supervised classification were that:

- Supervised classification is inherently limited by the information that can be inferred from the training data [18]. Meaning that, the accuracy and the representativeness of the training data, and also the distinctiveness of the classes must be taken into account. This tends to be a problem when dealing with large amounts document corpora, when no previous in-depth knowledge about the documents is assumed.
- Some documents tend to overlap, even when belonging to different categories. Such situations are quite common when working with documents with an average of 3.500 words each. In general, text classification is a multi-class problem (more than 2 categories). Training supervised text classifiers requires large amounts of labelled data whose annotation can be expensive [19]. A common drawback of many supervised learning algorithms is that they assume binary classification tasks and thus require the use of sub-optimal (and often computationally expensive) approaches such as one vs. rest to solve multi-class problems, let alone structured domains such as strings and trees [20].

- Labelling such documents manually beforehand is not a trivial task and may affect adversely the training set of the classification algorithm. Our intention is to reduce as far as possible human intervention in the classification task and also to scale up our approach to hundreds of scientific publications.
- The goal of the assessment is to evaluate if the semantic enrichment process improves the similarity level among documents, even when such documents were not considered similar using purely statistical approaches but, indeed, they are in fact similar from a semantic perspective.

4.1.4 Name-Entity Recognition Process

The Name-Entity Recognition (NER) process tries to locate and classify atomic elements in text into entity categories, such as names of persons, organizations or locations. This process aims at easing the mapping of names and locations into Cyc concepts, by classifying a text portion as a name or location.

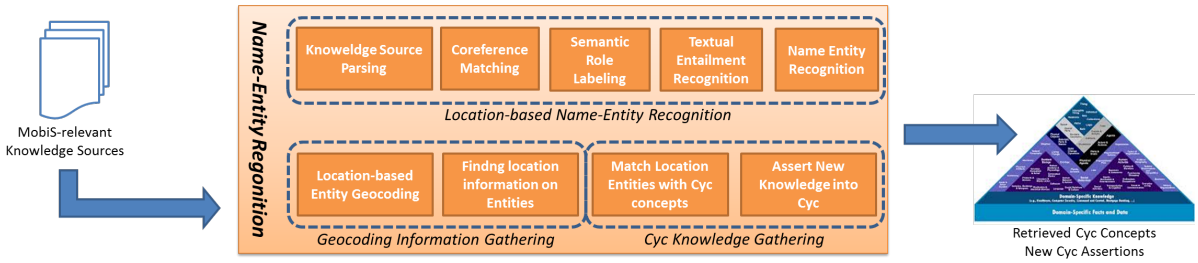

Figure 8. Name-Entity Recognition Process

This process uses a hybrid approach to locate entities in free-text, which uses two different but complementary Name-Entity Recognition tools: The Stanford CoreNLP Tool [21] and the Alchemy Entity Extraction API [22].

The Stanford CoreNLP provides a set of Java-based natural language analysis tools which can take raw English language text input and give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, and indicate which noun phrases refer to the same entities. Stanford CoreNLP is an integrated framework, which make it very easy to apply a bunch of language analysis tools to a piece of text. The Stanford CoreNLP's name-entity recognizer tool [23] provides three different classifiers, namely, a 4 class model trained with CoNLL training set, a 7 class model trained with MUC, and a 3 class model trained with both data sets for the intersection of these sets.

- 3 class - Location, Person, Organization
- 4 class - Location, Person, Organization, Misc
- 7 class - Time, Location, Organization, Person, Money, Percent, Date

Initial tests performed with the different classifying models prove that, for location search, the 4class CoNLL model is the best choice. As an example, the following text, taken from a Swedish traffic news feed, was used to test the three models:

"*Roadwork, a lane off. Route 222 on Stadsgardsleden between Slussen and Stadsgard terminal, towards Nacka. Approximately 40 meters of public lane off.*".

The CoNLL model was the best in this test, with the following result:

```
Roadwork/LOCATION,/O a/O lane/O off/O./O Route/O 222/O on/O Stadsgardsleden/MISC
between/O Slussen/LOCATION and/O Stadsgard/LOCATION terminal/O,/O towards/O
Nacka/LOCATION./O Approximately/O 40/O meters/O of/O public/O lane/O off/O./O
```

Despite the bad classification of the word “roadwork” as a location, all the other locations were caught by the classifier. Hence, the NER process uses this classifier to perform entity search.

The AlchemyAPI provides easy-to-use facilities for extracting the semantic richness from any text content: Post (upload) any content directly to our service for analysis. Posted content is analysed to detect the primary document language, and named entities are automatically extracted. These API calls may be utilized to process posted (uploaded) text content. This API also employs a sophisticated 'entity disambiguation' mechanism to resolve detected companies, locations, and people to a unique "instance”.

The NER process uses both APIs in order to find all entities in a particular text and disambiguate between entities, because one may identify some entities that the other cannot, and vice-versa. For instance, the error on the word “roadwork” may be solved if the results from Stanford’s CoreNLP API and Alchemy API are compared.

Furthermore, this process uses the Nominatim Geocoding API [24] to identify the nature of the location that is classified by the above API’s. When a location is tagged, the process uses Nominatim Geocoding API to find the exact type of place associated to that location (road, avenue, highway, neighbourhood, city, etc.), by querying the API with the name of the location found in the text, making locations easier to map to Cyc concepts or to assert directly into Cyc. A possible Nominatim result, represented in XML format, could be:

```
<searchresults timestamp="Sat, 07 Nov 09 14:42:10 +0000" querystring="135
pilkington, avenue birmingham" polygon="true">
   <place
    place_id="1620612" osm_type="node" osm_id="452010817"
    boundingbox="52.548641204834,52.5488433837891,-1.81612110137939,
    -1.81592094898224"
    polygonpoints="[['-1.81592098644987','52.5487429714954'],['
    -1.81592290792183','52.5487234624632'],...]"
    lat="52.5487429714954" lon="-1.81602098644987"
    display_name="135, Pilkington Avenue, Wylde Green, City of Birmingham,
    West Midlands (county), B72, United Kingdom"
    class="place" type="house">
      <house>135</house>
      <road>Pilkington Avenue</road>
      <village>Wylde Green</village>
      <town>Sutton Coldfield</town>
      <city>City of Birmingham</city>
      <county>West Midlands (county)</county>
      <postcode>B72</postcode>
      <country>United Kingdom</country>
      <country_code>gb</country_code>
   </place>
 </searchresults>
```

Information gathered through both processes (NER + Geocoding) will be used not only to find the location-related concepts which already exist under Cyc KB, but also to create new assertions about these locations with the new data available, such as location coordinates, city, country, etc, and add those locations as new Cyc concepts.

4.2 Pattern based Knowledge Extraction from unstructured text

This section describes knowledge extraction from unstructured text adapted to the traffic domain. The result of this process are the facts and concepts connected to traffic domain expressed in CycL assertions. For example, the sentence

"A lane was closed on South River Road."

can be expressed with the following CycL expression:

```
(#$thereExits ?EVENT
   ($#thereExists ?LANE
      (#$and
         (#$isa ?EVENT #$ObstructionEvent)
         (#$isa ?LANE ? #$LaneOfRoadway)
         (#$objectFoundInLocation ?LANE #$SouthRiverRoad)
         (#$objectActedOn ?EVENT ?LANE)
```

The extracted CycL expressions consist of concepts from CycKB and MobiS ontology extension. We further developed and adapted a tool called *PatternRules* to semi automatically extract CycL expressions from documents in the traffic domain.

We will shortly present the tool and how it was adapted to be used in the MobiS scenario. Slightly different version of PatternRules is presented in [25].

PatternRules is a tool used to extract structured knowledge from unstructured text that allows the user to semi-automatically construct pattern rules, which are used to construct the semantic expression for a fact in the text. This is done in macro-reading fashion [26], where the goal is to extract a large collection of facts from all documents, in contrast to extract every fact from a document. CycL expressions are constructed for facts in traffic documents.

4.2.1 Pattern rules

Pattern rules consist of lexical pattern and logical pattern (See Table 9). Lexical patterns consist of fixed words and placeholders. When lexical pattern is applied, placeholders become replaced by textual phrases. The replacing phrases must comply with the type of the placeholder, for example only phrases that are recognized as locations can replace the *[Location]* placeholder. We will call the replacing phrases - arguments of the lexical pattern. Lexical patterns may match with several fractions (e.g. sentences or shorter phrases) in the collection of documents. Logical patterns consist of concepts from the knowledge base and placeholders. When a rule is applied, the placeholders from the logical pattern become replaced by concepts, which represent arguments of the lexical pattern. In the example in Table 9, bridge Pont Neuf and river Siene are arguments of the lexical pattern.

	Lexical part	***Logical part***
Pattern	[Location] is a bridge over [Location]	(#$spansOver [Location] [Location])
Example	Pont Neuf is a bridge over Siene	(#$spansOver #$PontNeuf-Bridge #$SieneRiver)

Table 9 Example of pattern rule

4.2.2 Constructing pattern rules using PatternRules

In this section, we will present the process of making pattern rules using PatternRules. The user first selects the document, from which he would like to extract facts. Then, he selects the fraction of text, for which he would like to construct a pattern rule. In the generalization step, the user creates the lexical pattern by generalizing the selected text. The system will suggest to the user concepts from the knowledge base, which he can use to make the logical pattern. Newly created rules can be applied to the whole set of documents to obtain a list of CycL expressions, which are ready to be added to the knowledge base. We will now present each step of the process in more details. The graphical user interface (GUI) of the system is depicted in Figure 9.

Figure 9 the GUI of PatternRules

4.2.2.1 Document selection

The user selects the language (if there are documents from more than one language), and the number of the document (see top of Figure 10). The selected document will appear in the document panel (see right hand side of Figure 9)

4.2.2.2 Lexical pattern creation

The user selects the desired fraction of text and drags it to the lexical pattern panel. The interface for making lexical pattern will pop up (see Figure 11). For each token (word), the user selects the layer of generalization. There are several layers available for each token, e.g., plain word, lemma, part-of – speech tag, named-entity. If the plain word or lemma layer is selected, then the token will become a fixed word. If any of the other layers are selected, than the word will become a placeholder with the selected type. If a token is selected as an argument (last option), than it becomes a placeholder with no specified type. If consecutive tokens are chosen as arguments, they are grouped into a single placeholder. When all tokens are assigned a layer, the lexical pattern will appear in the lexical pattern panel (Figure 10). Every placeholder ends with the number, which is used to connect lexical and logical placeholders. It is clear that, the newly created lexical pattern matches the selected text, since it is the generalization of it. However, it may also match other fractions in other documents. The number of matches (frequency above the lexical pattern panel - Figure 10) serves as a guideline of how many new CycL assertions will the rule produce. Lexical pattern with higher frequencies are preferred.

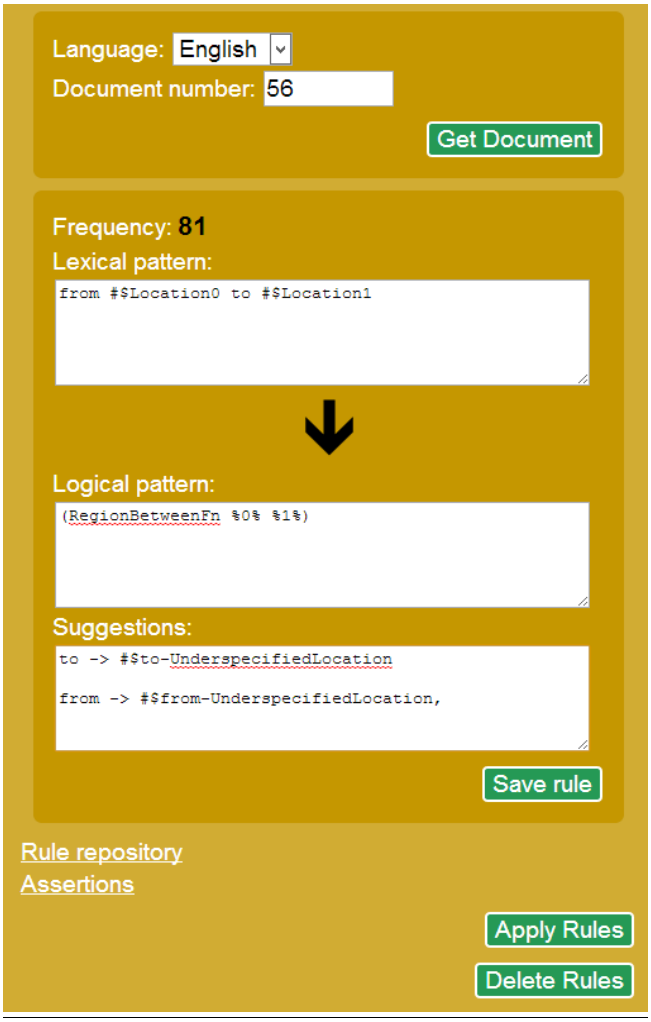

Figure 10 the main panel of the GUI

4.2.2.3 Logical pattern creation

The user creates the logical pattern in the logical pattern panel (Figure 10). The logical placeholders must be denoted by *%k%*, where k represents the number of the lexical placeholder that is connected to logical placeholder. The system will propose a generic logical pattern, like *(#$predicate %1% %2%)*. The user then modifies the initial logical pattern to make a final logical pattern that will produce valid CycL expressions. The system proposes concepts that may be used to create logical patterns in the suggestions panel (Figure 10). The suggestions save time that would be spend to browse the knowledge base to find the right concept. The proposed concepts are denoted by fixed words in the lexical pattern. A fixed word can denote many concepts. Therefore, the user must choose the right one or concept disambiguation is used to select the right concept.

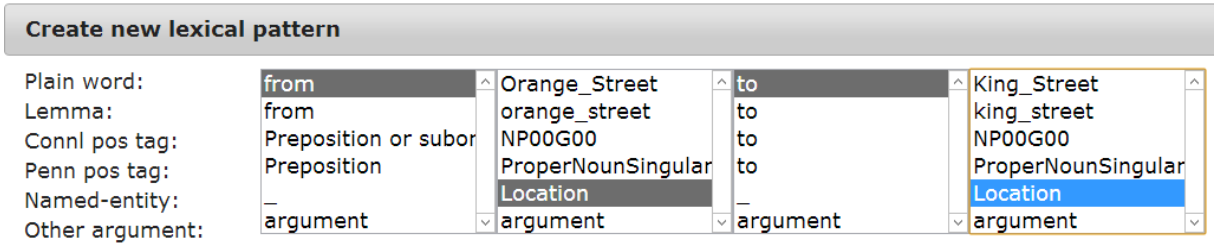

Figure 11 the lexical pattern construction interface

4.2.2.4 Applying rules

After the rule is finished, it is applied to the current document. The fractions of document that match the rule are highlighted in *green* (see document panel in Figure 9). If the user clicks, *Apply Rules* button, then all rules that have not been applied yet, are applied on the whole collection of documents. The list of all CycL expressions that have been produced can be seen by clicking the *Assertions* link.

Pattern rules can also be nested. Therefore, one rule can be used to translate the argument of the other. For example, "35 mph" is the phrase that matches the [Speed] placeholder in lexical pattern "speed limit is reduced to [Speed]". A CycL expression must also be provided for "35 mph" to fully translate the whole phrase. A rule like *"[Number] mph"* → *(#$MilesPerHour [Number])* can be used to translated the [Speed] arguments.

4.2.3 Adaptation

PatternRules was adapted and improved for better results in the traffic domain.

Initial collection of general domain documents from IJS newsfeed were classified using the algorithm from Section 4.1.3, to obtain a collection of traffic documents, which are relevant for MobiS scenario. The classification has two benefits. User does not need to traverse documents in PatternRules to find the relevant document. The other benefit is that the collection of documents is smaller. Therefore, the application of rules is much faster. In addition, the number of produced CycL expressions is not significantly smaller than on the whole collection, since the pattern rules are made for the specific domain.

Named-entity recognition is crucial for pattern matching. For instance, the bridge Pont Neuf must be correctly recognized and classified as a location, so that the lexical pattern from the example in Table 9 matches the sentence from the same example. Especially, location recognition is vital, since locations are prevailing type of named-entities in the traffic documents. PatternRules adapted for MobiS uses Name-Entity Recognition Process from the Section 4.1.4 to detect and link named-entities to CycKB, thus improving the precision and recall of the pattern rules application.

The CycL concept suggestions described in Section 4.2.2.3 are improved by the concept disambiguation mechanism described in Section 4.1.2.2.1. If there is more than one concept for a fixed word that would be otherwise suggested, the concept disambiguation selects the one, which is the most appropriate for the traffic domain.

4.2.4 Result of knowledge extraction

We constructed several pattern rules on the collection of 2000 traffic documents. Examples of pattern rules are presented in Table 10. The frequency of the rule reflects how many matches the lexical pattern of the rule has. The first two and the last two pattern rules can be nested, so that the first rule of the pair is nested inside the second. The last rule is grammatically suspicious, but occurs quite frequently in documents. For instance, in sentence "Daytime lane closures from 9 a.m. to 4 p.m. between Old Moultrie road and Interstate 95 to repair sections of damaged sidewalk."

Lexical pattern	***Logical pattern***	***Frequency***
[Number0] mph	(#$MilesPerHour [Number0])	175
speed limit is reduced to [Argument0]	(#$thereExists ?EVENT (#$and (#$isa ?EVENT #$SpeedLimitReduction) (#$reductedSpeed ?EVENT [Argument0]))	19
between [Location0] and [Location1]	(#$RegionBetween [Location0] [Location1])	1345
lane closures [Time0] [Location1]	(#$thereExists ?EVENT (#$and (#$isa ?EVENT #$LaneClosure) (#$dateOfEvent ?EVENT [Time0])	87

(#$locationOfEvent ?EVENT [Location1])

Table 10 Examples of pattern rules

5 Conclusions

The deliverable D4.2 MobiS Knowledge Capture Services, in the scope of WP4 Reasoning and decision function, describes the methods and approaches for knowledge acquisition from expert users, from the crowd as a special case, and natural language texts. On top of these methods a few prototype applications are being built. These prototypes are all somehow already connected to broader MobiS infrastructure, especially via the shared Knowledge Base (described in the D4.1 MobiS knowledge base), where the acquired knowledge can be queried or used in some other way by the different parts of the MobiS platform. Taking this into an account, the somehow successful implementations of the prototypes using the Knowledge Capture services, shows that this deliverable validates and continues the work done in D4.1, D2.2 and other components of the MobiS project.

The initial tests and evaluations of the prototype applications, shows that there is at least some implication of the commercial interest of such tools, especially in the direction of call centres and crowd sourced knowledge acquisition, which will be further explored and validated by the WP8.

Considering all this, the approaches taken in the D4.2 seems as a good direction and contribution to the overall MobiS project.

6 Annex

6.1 CALL CENTRE KNOWLEDGE ACQUISITION AND DECISION SUPPORT PROTOTYPE

This is the paper describing the Knowledge Capture in call centres. The abstract is in the following chapter, but the full paper can be found as a file *"CC_KnowledgeAcquisition.pdf"*.

6.1.1 Abstract

The purpose of this paper is to present the approach to knowledge acquisition and computer reasoning support in a call center environment. The common problem in such environments (especially in technical call centers) is that the call operators are usually just the interface to the experts inside the company. They often lack the detailed technical knowledge in the field of study, especially when they are newcomers who just started with their job. For this reason we developed the expert system (ES) that is able to obtain needed expertise from technical staff and it is able to assist less technically versed operators to provide feedback to customers and acquire the knowledge needed to fix the particular customer problem. The prototype implementation of the ES was built for the national roadside assistance call center which is focused on car fault diagnosis.

6.2 Knowledge Base Approach for Developing a Mobile Personalized Travel Companion

This paper is describing some of the methods from the chapter 3, which are jointly with some of the WP3 approaches as well implemented in the prototype mobile application. Abstract is in the next sub-chapter and the full paper is attached as file *"PersonalizedTravelCompanion.pdf.*

6.2.1 Abstract

Nowadays with the proliferation of a panoply of smartphones and tablets on the market, almost everyone has access to mobile devices, which offers better processing capabilities and access to new information and services. Taking this into account the demand for new and more personalized services which aid users in their daily tasks is increasing. Current navigation systems lack in several ways in order to satisfy such demand, namely, accurate information about urban traffic in real-time, possibility to personalize the information used by such systems and also the need to have a more dynamic and user friendly interaction between device and traveler. In short, the current navigation systems were not designed to be intelligent enough to communicate their needs effectively to humans, and they are not intelligent enough to understand humans' attempts to satisfy those needs. This paper presents an approach for developing a personalized mobility knowledge base to be used by a mobile travelling companion application, whose conversational interactions crowd source information about the world for the purpose of providing highly specific assistance and recommendations to the user. The work presented here, is still part of on-going work currently addressed under the EU FP7 MobiS project. Results achieved so far do not address the final conclusions of the project but form the basis for the formalization of the MobiS domain knowledge along with the MobiS data models.

7 References

[1] M. Witbrock, D. Baxter, J. Curtis, D. Schneider, R. Kahlert, P. Miraglia, P. Wagner, K. Panton, G. Matthews e A. Vizedom, "An Interactive Dialogue System for Knowledge Acquisition in Cyc," em *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[2] D. Lenat, "A Large-Scale Investment in Knowledge Infrastructure," *Communications of the ACM,* n.º 38.

[3] D. Baxter, B. Shepard, N. Siegel, B. Gottesman e D. Schneider, "Interactive Natural Language Explanations of Cyc Inferences," em *AAAI 2005: International Symposium on Explanation-aware Computing*, 2005.

[4] Jozef Stefan Institute, "IJS Newsfeed," 2012. [Online]. Available: http://newsfeed.ijs.si/. [Acedido em 2013].

[5] Trafikverket, "Datex II version 2.0," 2013. [Online]. Available: http://www.trafikverket.se/Om-Trafikverket/Andra-sprak/English-Engelska/The-Traffic-Information-Web-Site-Laget-i-trafiken/This-is-Datex/Technical-pages-for-subscribers/Datex-II-version-20/. [Acedido em 2013].

[6] Twitter, Inc., "Twitter REST API Documentation," 2013. [Online]. Available: https://dev.twitter.com/docs/api/1.1. [Acedido em 2013].

[7] M. Rajman e R. Besançon, "Text Mining - Knowledge extraction from unstructured textual data," 1998.

[8] Rapid-I GmBH, "RapidMiner," 2011. [Online]. Available: http://rapid-i.com/content/view/181/190/. [Acedido em 2011].

[9] K. S. Jones, "A Statistical Interpretation of Term Specificity and its Application in Retrieval," *Journal of Documentation,* vol. 60, n.º 5, pp. 11-21, 2004.

[10] P. Figueiras, R. Costa, L. Paiva, R. Jardim-Gonçalves e C. Lima, "Information Retrieval in Collaborative Engineering Projects – A Vector Space Model Approach," em *Knowledge Engineering and Ontology Development Conference 2012*, Barcelona, Spain, 2012.

[11] Cycorp, "Cyc KB," 2013. [Online]. Available: http://www.cyc.com/kb. [Acedido em 2013].

[12] R. Costa, P. Figueiras, L. Paiva, R. Jardim-Gonçalves e C. Lima, "Capturing Knowledge Representations Using Semantic Relationships," em *The Sixth International Conference on Advances in Semantic Processing*, Barcelona, Spain, 2012.

[13] S. Li, "A Semantic Vector Retrieval Model for Desktop Documents," *Journal of Software Engineering & Applications,* no. 2, pp. 55-59, 2009.

[14] D. Ramachandran e P. Reagan, "First-Orderized ResearchCyc: Expressivity and Efficiency in a Common-Sense Ontology," em *Papers from the AAAI Workshop om Contexts and Ontologies: Theory, Practice and Applications*, Pittsburg, Pennsylvania, 2005.

[15] P. Drineas, A. Frieze, R. Kannan, S. Vempala e V. Vinay, "Clustering Large Graphs via the Singular Value Decomposition," *Machine Learning,* pp. 9-33, 2004.

[16] M. Meilă, "The uniqueness of a good optimum for K-means," em *International conference on Machine learning* , Pittsburgh, 2006.

[17] J. MacQueen, "Some methods for classification and analysis of multivariate observations," Berkeley, 1967.

[18] M. Nagarajan, A. Sheth, M. Aguilera, K. Keeton, A. Merchant e M. Uysal, "Altering Document Term Vectors for Classification - Ontologies as Expectations of Co-occurrence," em *16th international conference on World Wide Web*, Alberta, 2007.

[19] S. Dumais, J. Platt, D. Heckerman e M. Sahami, "Inductive learning algorithms and representations for text categorization," em *international conference on Information and knowledge management*, Washington, 1998.

[20] A. Subramanya and J. Bilmes, "Soft-supervised learning for text classification," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Honolulu, Hawaii, 2008.

[21] J. R. Finkel, T. Grenager e C. Manning, " Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling," em *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, University of Michigan, USA, 2005.

[22] Alchemy API, Inc., "Alchemy API - Homepage," 2013. [Online]. Available: http://www.alchemyapi.com/.

[23] Stanford University, "Stanford's NLP Name-Entity Recognition Tool," 2006. [Online]. Available: http://nlp.stanford.edu/software/CRF-NER.shtml.

[24] OpenStreetMap Foundation, "Nominatim Geocoding API," 2013. [Online]. Available: http://wiki.openstreetmap.org/wiki/Nominatim.

[25] Xavier Carreras, Lluís Padró, Blaž Fortuna, Janez Starc , "Early Semantic Triple Graphs Merging Prototype Deliverable D4.2.1," XLike project, 2012.

[26] Mitchell, Tom M and Betteridge, Justin and Carlson, Andrew and Hruschka, Estevam and Wang, Richard, "Populating the semantic web by macro-reading internet text," em *{The Semantic Web-ISWC 2009*, 2009.