

Time event discovery from discourse representation structures and schedule ontology instantiation



Research Report 2006

by: Oscar Medina Duarte

Advisors: Macq Benoit, Olga Vybornova.

Special thanks: Johan Bos.

Abstract.- This report presents an algorithmic approach for time event references discovery in a discourse representation structure. We present a preliminary implementation of the object oriented model of the DRS and the algorithm proposed. As well as some discussion on how to implement a schedule ontology that can be instantiated from the output of our work.

Table of Contents

- Introduction.....3
- Appropriate Working Space.....4
- Object Oriented DRS design.....5
 - Implementation.....6
- Schedule discovery in DRS.....6
 - Argument discrimination.....7
 - Heuristics.....7
 - Time and Date and Action unification.....8
 - Schedule Discovery Algorithm (SDA).....9
 - Implementation.....10
- Ontology Design.....10
- Results.....11
 - Conclusion.....13
- Appendix A – UML design of the Object Oriented DRS representation.....14
- Appendix B - Ontology's Design.....15
- Appendix C - DRS's XML semantics.....16
- Appendix D – List of annoyances on the ccgparser and/or ccg2drs software.....17
- Appendix E - List of files.....17

Introduction

Houses are changing their nature, from static wooden structures to dynamic and technology enabled living and working environments. In a perfect world, this new kind of environments would actively assist and interact with their inhabitants to provide an enhanced living and working experience. This Ambient Intelligence (AmI) is expected to work autonomously and may interact with the inhabitants spontaneously but not invasively. Such system should keep a base of knowledge about all aspects of its environment, this may allow the system to take an intelligent part on the day to day living.

Schedule discovery is a potentially important part of an AmI multimodal infrastructure, it is intended to assist, increase autonomy, improve quality of life, minimize health risks of elderly and mild cognitive impaired people. This work focuses on schedule discovery through language analysis, in particular, by extracting information from discourse representation theory's (DRT) structures (DRS) and instantiating a schedule ontology. Schedule discovery pretends to be part of a proper AmI platform providing part of the framework required for an activity reminder module, which would help people remember and perform essential everyday tasks by compensating decreased human memory capabilities.

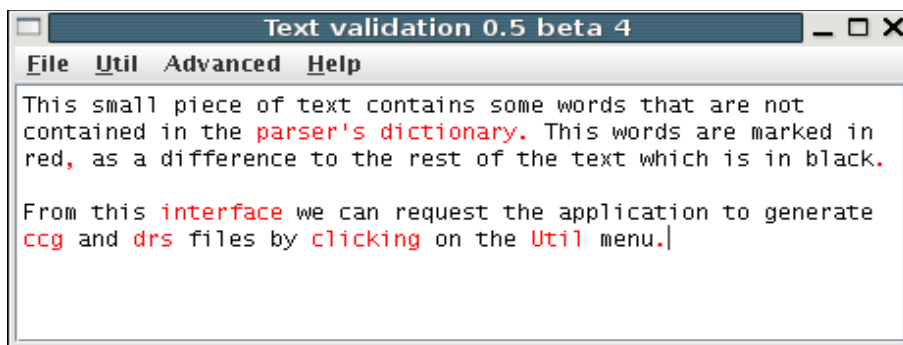
Discourse Representation Structures are assumed to be available. In this case, we used the CCG-parser C&C (candc) by Steve Clark and James Curran and the compositional DRT semantics component that generates a well formed DRS developed by Johan Bos (cgg2sem). Currently there is no complete implementation of DRS available, in particular for the case of presupposition resolution which is discussed in <Deemagarin Amarin's work,2006>.

Appropriate Working Space

The available implementation of DRSs works with prolog but it only has a command line interface, which is not friendly enough, even for an experienced user, therefore is not very practical for an intensive use and is not ready for integration with other language platforms like Java or Python.

Another restriction is the fact that the parser has a defined list of valid words, which means that its behavior is unpredictable when words out of the list are part of the input.

In order to solve this problem and accelerate our experimentation with the software, a graphical user interface (GUI) was implemented. From the user point of view, it allows access to the DRT/DRS software without typing more than the text that is required. The typing is assisted by a highlighted mode to check for words out of the parser's list. This last feature gives an intermediate feel between a spell checker and an interface development environment (IDE).



From the lower level point of view, a Java API was developed to provide convenient access to the parser and to the DRS software, which is done by imitating the required user calls to the software in Java.

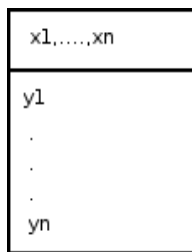
Object Oriented DRS design

The main intention of having a proper object definition of DRS is to be able to create a navigable representation that easily allows the implementation of searching algorithms in a high level language.

Discourse Representation Structures can be seen as pictures that contain a representation of the information in discourse and the relationships between its different parts. DRS are pairs consisting of a finite set of discourse referents and conditions and represented as boxes. Complex conditions can be built out of boxes using the connectives \neg , \vee , and \rightarrow .

So here a formal definition of what a DRS is:

1.- if x_1, \dots, x_n are discourse referents ($n \geq 0$) and y_1, \dots, y_n are conditions then:



is a DRS.

2.- If R is a relation symbol of arity n , and t_1, \dots, t_n are terms, then $R(t_1, \dots, t_n)$ is a condition.

3.- If t_1 and t_2 are terms then $t_1 = t_2$ is a condition.

4.- If B is a DRS, then $\neg B$ is a condition.

5.- If B_1 and B_2 are DRSs, then $B_1 \vee B_2$ is a condition.

6.- If B_1 and B_2 are DRSs, then $B_1 \rightarrow B_2$ is a condition.

7.- Nothing else is a DRS or a condition.

The most important aspect in the design of such an object oriented specification is having the ability to access any part of the DRS starting from a word in the original utterance and from such word, finding all the referents and conditions. So in practice, it would not be so useful to specify a formal minimalistic DRS, instead we can create enough data structures and properties around it to be able to do higher level operations.

Also, for practical matters, the XML specification of DRS given by ccg2drs's output was taken as a starting model for the object oriented specification, which leads to the object oriented specification shown in the UML diagram of appendix A. Although, in the last days, a new and more precise XML specification its being developed together with Johan Bos which would accelerate the loading time and prevent ambiguity's.

Implementation

After a proper design of the object oriented representation of the DRS, the code was generated, at a first stage using Poseidon for UML Community Edition automated code generation. From this step, most of the files and structure of the DRS were created and in a second stage, the actual implementation of the classes took place. The internal architecture of each class is not dealt in this document since it is a very extensive subject.

In order to load the DRS representation in XML, the dom4j XML API was used. This API has several advantages like its license, the incorporation with SAX and Xpath. This was the longest task of the implementation since there are many details and restrictions that had to be inherited from the DRS and from the XML DTD of the document to create a proper data structure.

Schedule discovery in DRS

There are different kinds of utterances that express time and scheduling. In this work, we are only interested in the most explicit ones, those that express an action (verbal expression) related to a time reference complement. We started by exploring the DRS implementation in hand with different very simple utterances that intuitively can be identified as schedule references.

Eleven different structures were examined in a first view, using 5 different verbs (55 utterances) and their DRS box representation compared. The different structures are:

<Verbal part>

I have to <Verbal part>

I have to <Verbal part> at four

I have to <Verbal part> at 16:00

I have to <Verbal part> at 16

I should <Verbal part> at 16

I would like to <Verbal part> at 16

Is important to <Verbal part> 16

I wish I could eat <Verbal part> 16

I want to <Verbal part> at 16

now I have to <Verbal part>

Where the verbal parts were the following: “eat”, “take my medicine”, “meet Olga”, “go to the bank” and “do the laundry”.

The results, in general, showed that the easiest way to access the information was through the preposition of time *at*, and that there is, in most cases, a very similar way to access the time reference portion and the verbal portion of the utterance, therefore some more DRSs were generated, but this time taking the prepositions of time into account.

The tested prepositions were : at, in, on, after, before, by, during, for, from ... to, from...till, from...until, past, to, up to and within, some other prepositions were not analyzed because they are applied to the past, or their meaning in the context of schedule discovery depends strongly on the context. The prepositions that were not tested are: ago, since, till and until.

The utterances tested for the prepositions were:

<VB> at night	<VB> on the morning of September the 11th
<VB> at Easter	<VB> after school
<VB> at Home	<VB> after finishing the work
<VB> at 23 minutes past 6	<VB> before Christmas
<VB> at 23 minutes to 6	<VB> between Monday and Friday
<VB> in July	<VB> between the 11th of September and the 25th of December
<VB> in the morning	<VB> by Thursday
<VB> in a minute	<VB> during the holidays
<VB> in two weeks	<VB> for three weeks
<VB> on Sunday	<VB> from Monday to Wednesday
<VB> on the 25th of December	<VB> from Monday till Wednesday
<VB> on Good Friday	<VB> from Monday until Wednesday
<VB> on Easter Sunday	<VB> up to 6 hours a day
<VB> on my birthday	<VB> within a day

where <VB> := I have to do something

Argument discrimination

The first step is finding the arguments of the preposition, for some have more than two, such as *between*, for which the repeated type of argument is the one referencing time or space, etc. The idea here is to locate the arguments and determine if they are verbal, of time, spatial or other kind of argument.

In this case, the arguments are always Discourse referents. A finer grain description of this process would be to identify the kind of discourse referent we are dealing with. There are discourse referents that group several words together which is especially common in time and date referencing. To confirm that we are actually dealing with a time referent (see *timex* in appendix C) we can do different things like finding out if a word is inside a discourse referent set of words that contains a named entity referring to time (i.e. I-DAT) or if they are adverbs of time, etc.

Similarly, one of the arguments of the preposition has to be a verb or a verbal direct complement, whose detection is performed in a similar way as the time referent.

Heuristics

The first heuristic to apply to an argument is to try to find its lexical category. If it is a verb (VB), the argument is saved as the verbal part of the expression; if not, we try to find out if we are dealing with a more complex verbal structure.

Then, if it is a named entity (NE) of type I-DAT or its part of a *timex* structure or it is a known adverb of time or a combination of the above, we assume the argument is a reference of time. Sometimes the date numbers are identified as

some type of cardinality, although this is still a bug in the ccg2drs, we can see if they are followed by a NE of type I-DAT or some other time reference and have a good chance to identify them correctly as part of a time reference.

If the argument is a list of words and all of them are I-DAT NEs, it is a time reference. If some of the words of the list are I-DAT NEs and the rest of them are not verbs or direct complements of verbs, then all of them are a time reference.

If some of the words from the list are not time reference, they could be a verbal part or a verbal complement. In order to identify them, we apply the first heuristic to see if they are verbs. If they are not, then we check if they are patients of a verb; if they are, we can easily find the verb applied to them. When a verb is found, we are interested in finding all the semantic roles connected to it to have a complete image of the time event.

For a more robust solution to this classification problem, we could implement a case based reasoning solution, in which all the correctly identified cases form our set of known cases.

Time and Date and Action unification

Once we have found the verb we try to find its agent and patient semantic roles and a time reference, we put them together in another data structure to form a time event structure (TES). This TES is one primitive ontology that will allow a proper schedule ontology to be instantiated.

The reverse engineered version of this ontology looks like the image below.

TimeEventStructure
drModal : DiscourseReferent preposition : String timeReference : TimeReference verbalReference : VerbalReference
getDrModal() : DiscourseReferent getText() : String toString() : String getTimeReference() : TimeReference getVerbalReference() : VerbalReference setDrModal(drModal : DiscourseReferent) : void setTimeReference(timeReference : TimeReference) : void setVerbalReference(verbalReference : VerbalReference) : void

VerbalReference
agent : DiscourseReferent patient : DiscourseReferent verb : DiscourseReferent
<<create>> VerbalReference() getAgent() : DiscourseReferent getPatient() : DiscourseReferent getText() : String getVerb() : DiscourseReferent setAgent(agent : DiscourseReferent) : void setPatient(patient : DiscourseReferent) : void setVerb(verb : DiscourseReferent) : void

TimeReference
date : Date root : String timeRef : DiscourseReferent timeString : String
getRoot() : String getText() : String getTimeRef() : DiscourseReferent setRoot(root : String) : void setTimeRef(timeRef : DiscourseReferent) : void

Schedule Discovery Algorithm (SDA)

The most important part of a schedule discovery algorithm is the recognition of time events, once that step is performed, the next part of the work would be to solve locate these events in the real time, for instance, when expressions of desire or need are followed by a time referent that is relative to the current time or relative to some other well known event like Thanks giving day, Christmas eve, my birthday, or other previously recognized and stored schedule events like meeting the doctor, feeding the dog etcetera.

High level algorithm explanation:

```
Input: Original tagged text, DRS
Output: List of TimeEvent's : TimeEventStructure
1) find all prepositions of time in the original text: PREP

2) for each PREP, find all their rel conditions inside of the
   DRS. It doesn't matter how deep inside of the DRS: RELS
   2.1) for each RELS, extract its Argumet1 and Argument2
       // If no arguments, DRS may not be well formed
       2.1.1) for each argument (Arguments are Discourse
referents)
           2) if argument {is a time referent}
           3) Save it : TimeREF
               3.1) if TimeREF is a patient
                   2) We can find the verbal part too
           4) else if argument {is a verb referent}
           5) Save it : VerbREF
           6) Fill TimeEventStructure
       3) If TimeEventStructure is completely filled,
       4) add it to the list
       5) else if TimeVerbPair is incomplete
       6) ## Unknown condition: Find a proper policy
       7) else
       8) ## Unknown condition: We may not be dealing with
           a preposition.
3) Return list of TimeEventStructures
```

The Tagged text, is the original utterance with all its words marked by its lexical categories. The most important parts of this algorithm are the verb and time identifiers, which are described below.

```
Input: Discourse Referent : dr
Output: TimeREF

{IS A Time Referent} if:
1) dr is NamedEntity type I-DAT or
2) dr is a cardinality or
3) dr is contained in a list
   together with other time reference or
4) dr is an adverb of time or
```

This algorithm finds out if the given a discourse referent it's a time reference, and fills a representation structure. The second condition its derived from a bug in the ccg2sem software which identifies some date and time named entity's as some cardinality type.

```
Input: Discourse Referent
Output: VerbREF

{IS A Verb Referent} if:
  1) Its lexical category is VB
    1.1) If verb identified
      2) solve its agent and parient semantic roles.
```

Given a Discourse referent, find out if it is a verbal reference and recover its agent and patient semantic roles.

This algorithms can be improved by applying artificial intelligence techniques and by including all the semantic structures known for time event expressions.

Implementation

There two main methods to implement the proposed algorithm, they differ greatly in speed, but they differ also a lot in code understandability, the slower one but easy to read code (object recursive algorithm) was implemented because the purpose so far is to see how the algorithm behaves. The lack of performance of this algorithm resides mainly in the fact that requires a lot of object instantiations and string manipulation, which are two tasks that are not particularly fast in Java.

The option of implementing an iterative instead of recursive algorithm is good for implementation and deployment purposes since the usage of a good data base server to store and access utterances and DRSs is recommended because of its speed and autonomy. An important starting point for such algorithm would be a good relational design of the DRS.

Ontology Design

To specify the concept of this schedule, it is very important to have in mind that this schedule is meant to be used in a quite specific case. First, it is known to be part of a larger system, which will use it actively to monitor the accomplishment of tasks, trigger reminders, and other schedule administration aid. Secondly, it should be able to contain incomplete information to be kept for future use and completion, policies for managing incomplete ontology should be part of the above mentioned AmI and as such it is out of the scope of this document.

As this is not regular schedule ontology, some particularities are meant to be taken care of, like the inclusion of a task's status. For instance, to be able to know if a task should be in course, and/or if it already is in course, of if it was delayed, but must be completed.

To set a minimum notion of correctness on the ontology's design, a set of competency questions has been proposed to serve as a parameter of the minimum expressiveness required for the ontology.

- What is the nearest task to start?
- Is there any task that should have already been started?
- What is the last completed task?
- What is the nearest task to complete?
- What is the most recent added task?
- Which are the pending tasks for the next <N> <TIME> ?

The main ontology's class is the *Task* that contains a unique identifier, a name, which is a string extracted from the TES which says what is the task to perform, a status indicating the progress of the task (not yet started, started but not finished, finished, expired, etcetera...), creation time, importance field, whose meaning can vary slightly depending on the context of deployment, like the age of the person that uses it, its gender, its social roles, compromises and many other. The last attribute is a status modifier which indicates a default policy for the current instance's status, for instance, to indicate that a uncompleted task can be ignored or it should trigger an alarm, or to move the appointment forward in time until its completed or some other reactive measures to apply during the monitoring.

The *Appointment* class is a subclass of *Task* that contains the related temporal information which is a localized in time version of the *TimeReference* extracted from the DRS. Finally, all this is stored by the *Schedule* and *ListOfAppointments* classes, which for a high performance application should be implemented by a database.

Results

The time event discovery implementation was tested with the corpus described before. In general, with the current algorithm we were able to identify 74% of the time preposition events of the corpus, some of the unidentified ones are not propositions of time and some other are but they were not resolved completely. Here part of the output of unrecognized events accompanied by its utterance at top:

```
I would like to go to the bank at 16
[at] = arg1{
    @T: the postag: DT
    @V: the postag: DT
    Unrecognized case: i40006 : _G31489 ~ the }
arg2{
    Time reference: i40009 : _G31510 ~ 16
}
```

Human diagnosis : the bank is not a verb but is patient of go, go to the bank

Is important to go to the bank at 16

```
[at] = arg1{
  @T: Is postag: VBZ
  @V: Is postag: VBZ
  Unrecognized case: i41001 : _G44475 ~ Is }
arg2{
  Time reference: i41008 : _G44496 ~ 16
}
```

Human diagnosis: This is apparently not well represented by the DRS software.

I have to do something at home

```
[at] = arg1{
  @T: do postag: VB
  Verb reference: i58004 : _G9563 ~ do }
arg2{
  @T: Home postag: NNP
  Unrecognized case: i58007 : _G9695 ~ Home
}
```

Human diagnosis : This is not a time preposition, but it may have a context dependent meaning.

I have to do something in the morning

```
[in] = arg1{
  @T: do postag: VB
  Verb reference: i62004 : _G19551 ~ do }
arg2{
  @T: the postag: DT
  Unrecognized case: i62007 : _G19683 ~ the
}
```

Human diagnosis : This is a DT that points to an adverb of time, this should be implemented.

I have to do something after school

```
[after] = arg1{
  @T: do postag: VB
  Verb reference: i71004 : _G41820 ~ do }
arg2{
  @T: school postag: NN
  Unrecognized case: i71007 : _G41841 ~ school
}
```

Human diagnosis : School is not an adverb of time, but it is a reference to other contextual information that we could have stored in our schedule, future work should include way to solve this kind of references.

I have to do something after finishing the work

```
[after] = arg1{
  @T: do postag: VB
  Verb reference: i72004 : _G60060 ~ do }
arg2{
  @T: finishing postag: VBG
  Unrecognized case: i72007 : _G60263 ~ finishing
}
```

Human diagnosis : Same as the last one but a bit more complex structure, because “the work” can be ambiguous and may lead to an incorrect time event.

I have to do something up to 6 hours a day

```
[up] = arg1{
  @T: something postag: NN
  @V: something postag: NN
  Unrecognized case: i82005 : _G1911 ~ something }
arg2{
```

```
@T: something postag: NN
@V: something postag: NN
Unrecognized case: i82005 : _G1932 ~ something
```

}

Human diagnosis : This might not be properly represented by the DRS output.

Those were the most common errors and possible explanations, the most frequent one is the appearance of false time prepositions, which means that to solve it we have to be able to determine cases where the preposition is not a time preposition. The second most common, is related to the access to adverbs of time via a determiner like the or a, in this case we just have to do an extra level of semantic processing.

Once the time events were disclosed, we tried to reconstruct them in a human readable form, and we got the following result:

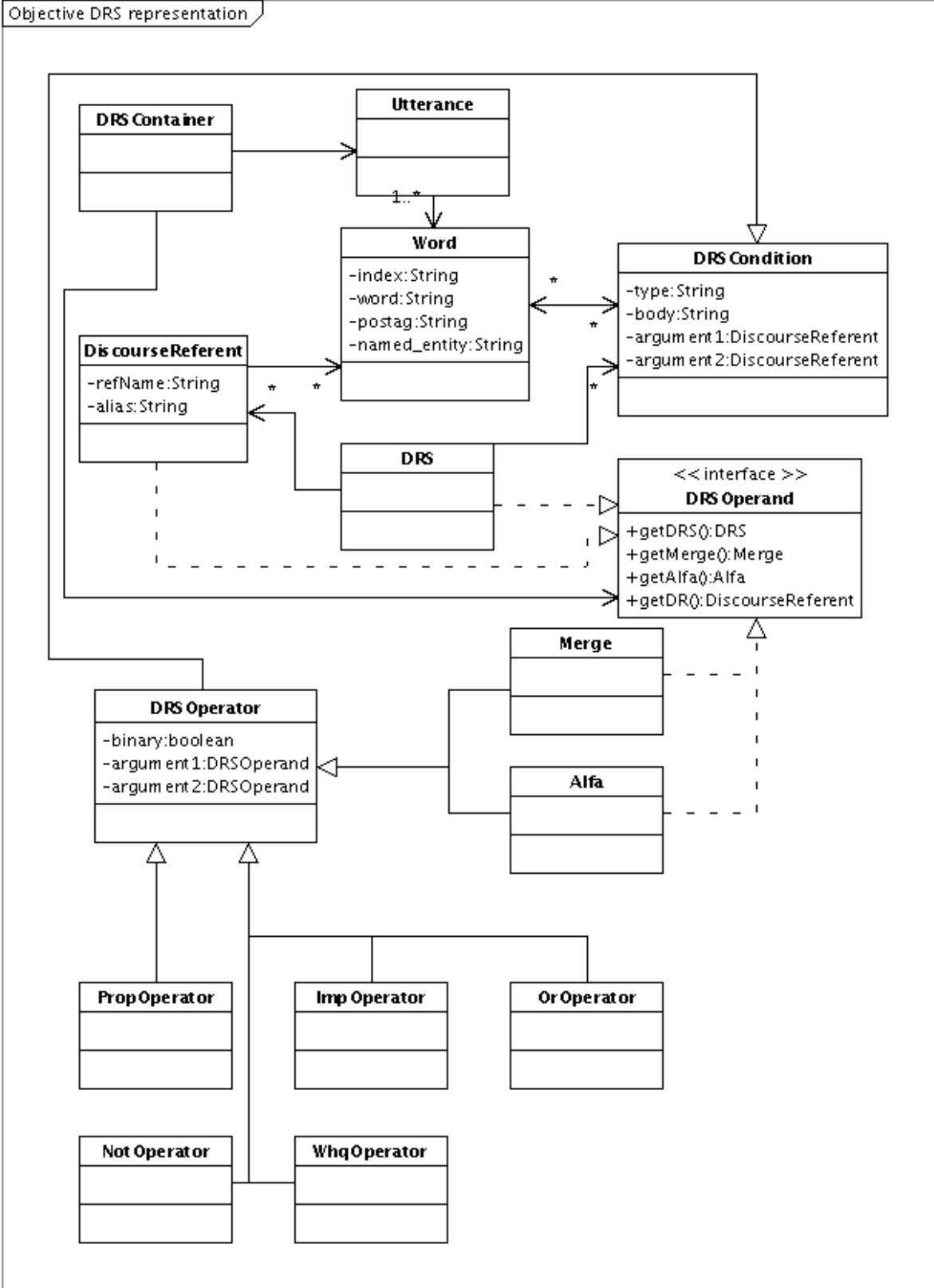
```
I eat <unknown Patient> at four
I take my at four
<unknown Agent> take my at 16
I do something at night
I do something to 6
I do something at 23 minutes
I do something in July
I do something in two weeks
<unknown Agent> do something on Sunday on something on Sunday
<unknown Agent> do something on the 25th of December on something on the 25th of December
<unknown Agent> do something on Good Friday on something on Good Friday
<unknown Agent> do something on Easter Sunday on something on Easter Sunday
<unknown Agent> do something on my birthday on something on my birthday
I do something before Christmas
I do something by Thursday
I do something for three weeks
I do something to Wednesday
I do something from Monday
I do something from Monday
I do something till Wednesday
I do something from Monday
I do something until Wednesday
```

From these results we can see that there is still some work to do regarding personal pronouns and agent identification.

Conclusion

The results and test of this work proved that event discovery from DRS structures is a problem that can be addressed in a quite precise way and with a logarithmic computational complexity, and since the problem is quite localized, its size doesn't grow a lot. With a proper implementation having performance in mind and good computational resources we can reach speed levels that may feel like real time to users.

Appendix A – UML design of the Object Oriented DRS representation



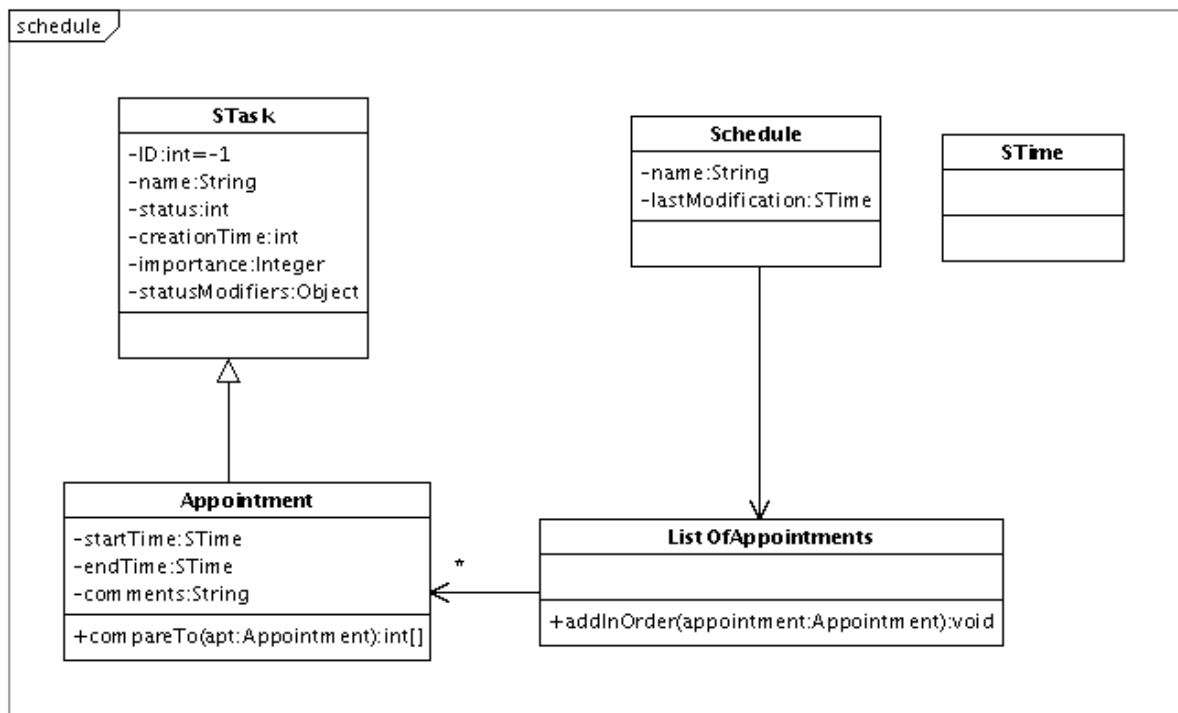
Appendix B - Ontology's Design

Name: Schedule

Keywords:

ID, Name, Status, Creation Time, Importance, statusModifiers, lastModification, startTime, endTime, comments.

UML Diagram



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Appendix C - DRS's XML semantics

This is an attempt to explain the semantics of the XML files generated by the software <of Johan> from a very high level point of view. These terms are taken from the *xdrs.dtd* file. No documentation about this semantics was found, therefore, all the information contained here was obtained by comparing dozens of different utterances

xdrs-output - Contains one or more xdrs elements. No attributes

xdrs - Defines all the parameters for a DRS. It contains words, postags, netags, drs, merge and alfa elements. Attributes: Unique ID.

drs - This is the actual DRS representation, it contains dr, timex, pred, card, rel, prop, not, or, imp, whq. No attributes.

merge - This is , it contains a pair of drs, merge or alfa. No attributes.

alfa - This is a relation of unresolved DRS, it contains a pair of drs, merge, or alfa. Attributes: type.

dr - It defines a discourse referent, it contains a name or ID. Attributes: index is a unique identifier.

pred - ... is this a predicate part of the DRS, it contains the name of the predicate. Attributes: index is a unique identifier, one argument.

not - Negation, contains drs, merge and alfa. Attribute: ID

or - Disjunction, contains a couple of drs or merge or alfa. Attribute: ID.

imp - Implication ?? contains a couple of drs or merge or alfa. Attribute: ID.

whq - wh clef, contain a couple of drs or merge or alfa. Attribute: ID.

prop - is this a proposition or conditional or what?. Contains a drs or merge or alfa. Attributes: ID, argument.

timex - Time reference (not completely implemented yet?) contains date or time. Attributes: ID, arg.

date - Date string in YYYYMMDD format.

time - ... time (incomplete specification?)

rel - Relationship, contains the name of a condition. Attributes: ID, arg1, arg2.

card - Cardinality, it indicates that the entry is a number, contains one of ge or eq (greater or equal and equal respectively). Attributes: ID, arg, type.

words - Contains one or more word. No attributes.

word - Contains a word from the utterance. Attributes: ID.

postags - Contains one or more postag. No attributes.

postag - Contains a lexical name (ie. VB, NN, NNP, CD etc...). Attributes: ID.

netags - Contains netags. No attributes.

netag - Named Entity (NE). Contains one of I-DAT, I-ORG, I-PER, I-LOC. Attributes: ID.

Appendix D – List of annoyances on the ccgparser and/or ccg2drs software

1) Some utterances are not processed but still they are counted, ie. from a list of 55 utterances (see Appendix B) two of them did not produce any output, but they were counted, that means that in the output, we had the output indexes from d1 to d50, d52 to d54. Utterances d51 and d55 missing, the example utterances are:

“I would like do the laundry at 16”

and

“now I have to do the laundry”

2) The timex structure does not seem to recognize all kind of date and time named entity's. An improvement on this point would significantly improve results and algorithm.

Appendix E - List of files

time_event_drs_20060623.pdf - This pdf file.

Parse Validator-20060623.tgz - GUI for the candc and ccg2sem programs.

jXDRS-20060623.tgz - Object oriented implementation of DRS model and time preposition resolution algorithm.

jXDRS-javadoc.tgz - Javadoc of the jXDRS API.