GENI IN FROLOG
08/03/2009

1. Geni inside Frolog's architecture

Frolog (Benotti, 2009) is a dialogue system that implements a text adventure game. It uses state-of-the-art tools for the most heavy loaded tasks (depicted in grey):

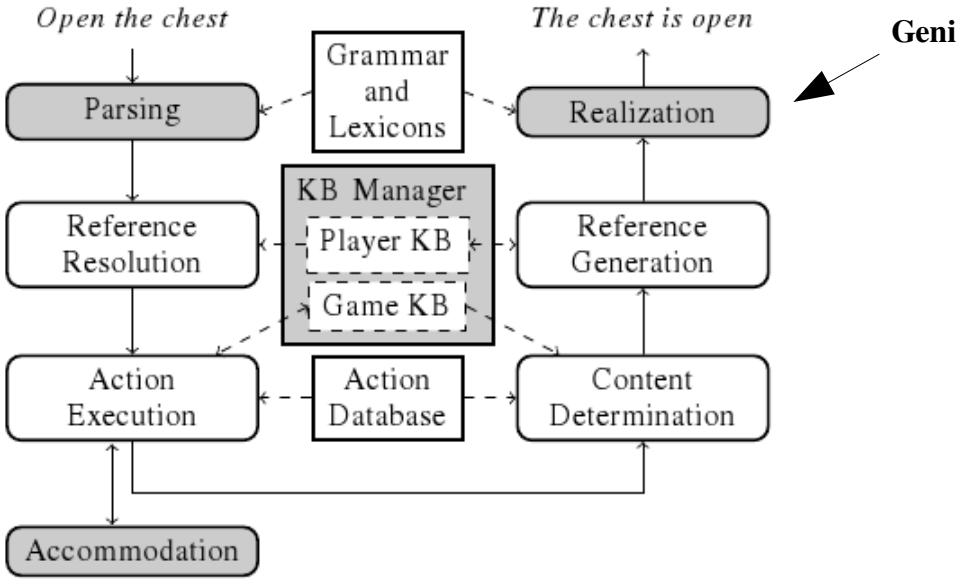
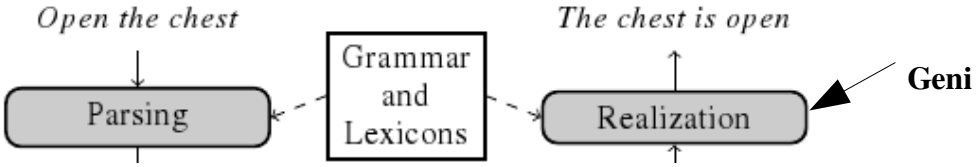


Figure 1: Architecture of Frolog

- For **parsing** it uses a Tree Adjoining grammar parser called Tulipa (http://sourcesup.cru.fr/tulipa/)
- For **knowledge base management** it uses the Description Logic reasoner RACERPro (http://www.racer-systems.com/)
- For part of the **dialogue management** it uses the artificial intelligence planner Blackbox (http://www.cs.rochester.edu/u/kautz/satplan/blackbox/)
- For **realization** it uses Geni (http://trac.haskell.org/GenI/wiki)

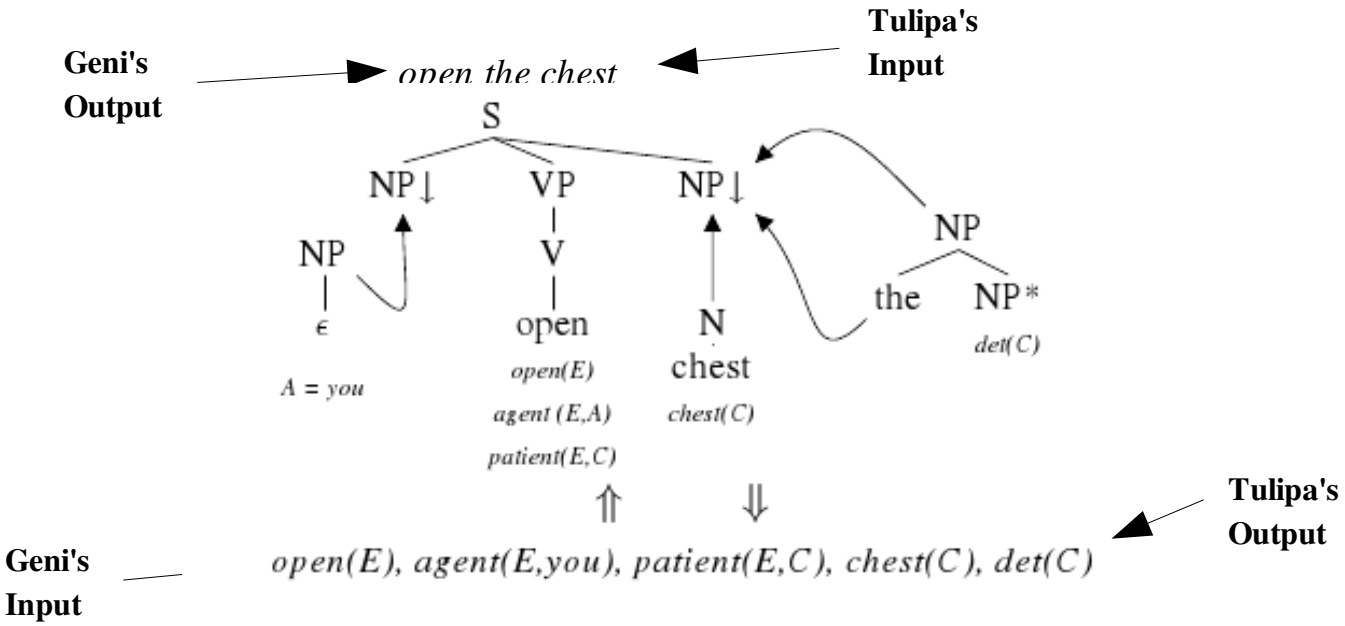
1.1 Parsing anf Realization in Frolog

The parsing and the realization modules use the same linguistic resources, which are loaded only **once** in Frolog when it starts, namely:



- a reversible grammar in the XMG grammatical formalism (Crabbe and Duchier, 2004) (http://wiki.loria.fr/wiki/XMG/Documentation).
- a lemma lexicon and
- a morphological lexicon

The parsing module performs the syntactic analysis of a command issued by the player, and constructs its semantic representation using the TAG parser Tulipa (Kallmeyer et al., 2008) (illustrated in the Figure 2 by ⇓).



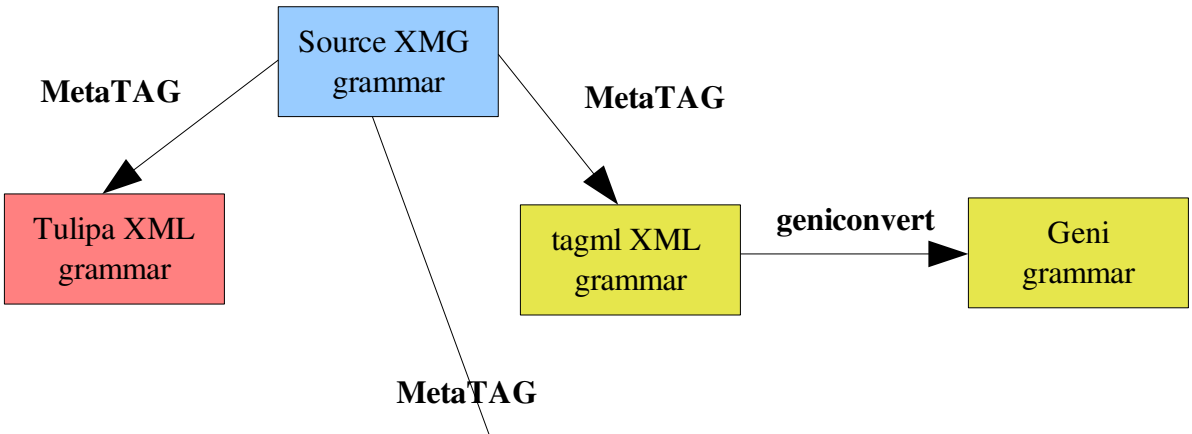
The realization module works in the opposite direction, verbalizing the results of the execution of the command from the semantic representation using the TAG surface realizer GenI (Gardent and Kow, 2007) (illustrated in the Figure 2 by ⇑).

2. The resources: A reversible grammar and lexicons

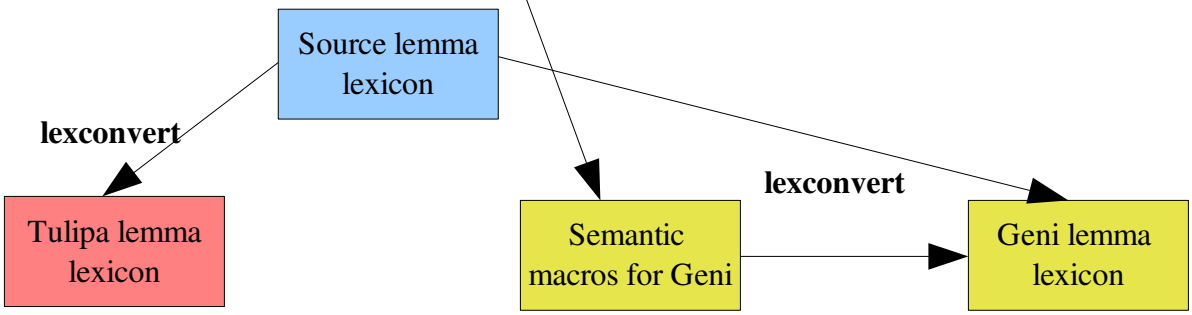
We say that Frolog uses a reversible **grammar** because the same grammar and lexicons are used for parsing and realization. However, the formats of the resources accepted by Tulipa and Geni are different.

A general architecture of the required conversions is:

For the grammar:



For the lemma lexicon:



These are the tools to automatically convert between formats:

- **MetaTAG** can be found at http://wiki.loria.fr/wiki/XMG/Documentation#MetaTAG
- **Lexconverter** can be found at http://wiki.loria.fr/wiki/LEX2ALL

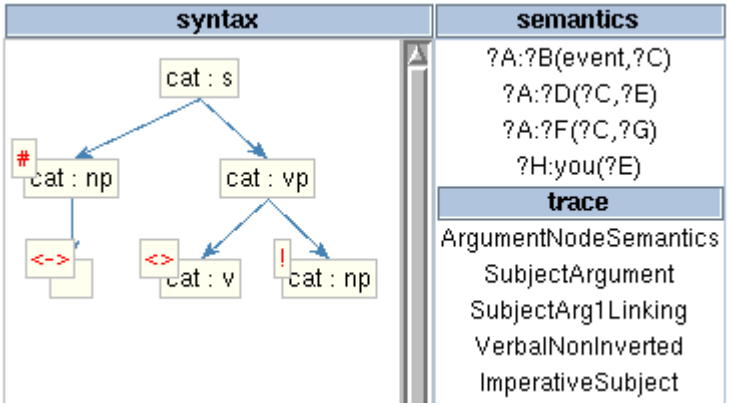
Note: The morphological lexicon conversion should work in a similar way but is not yet implemented.

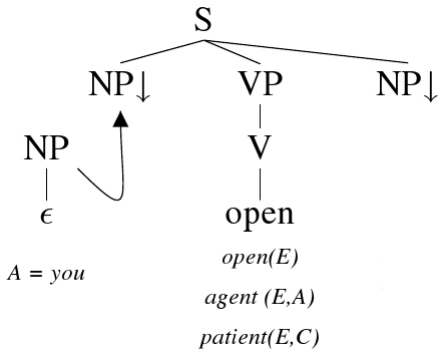
So the files that have to be designed by hand are only the ones in light blue: the **Source XMG grammar** and the **source lemma lexicon**.

2.1. The SOURCE grammar: An XMG grammar for English

- Frolog uses an XMG grammar for English called XMG-basedXTAG that was developed by Katya Alahverdzhieva <katya.alahverdzhieva@gmail.com> (a student of the Masters TAL).
- Documentation about the grammar can be found in http://svn.devjavu.com/katya/XMG-basedXTAG/.
  It's design is based on XTAG, a project to develop a wide coverage grammar for English at the UPENN (http://www.cis.upenn.edu/~xtag/)
- The XMG grammar specifies a TAG of around 500 trees and integrates a semantic dimension a la (Gardent, 2008).

TAG tree for transitive verbs n0Vn1 seen with MetaTAG GUI: "Same" TAG tree anchored in "open":





XMG specification of transitive verbs with semantics:

```
class  n0Vn1
{      binaryRel[];
       Dian0Vn1[]}


class binaryRel
declare  ?L0 ?Rel ?E ?X ?Theta1 ?Y ?Theta2
      {
       <sem>{L0:Rel(event,E);L0:Theta1(E,X);L0:Theta2(E,Y)}
             *=[label0=L0,rel=Rel,arg0=E,arg1=X,theta1=Theta1,arg2=Y,theta2 =Theta2]
       }


class  Dian0Vn1
{      dian0Vn1active[]}

class  dian0Vactive
import
       SubjectArg1Linking[]
{      active[] ; Subject[]}
```

Semantic class

Interface class

Syntactic classes

Note: The definition of the classes in italic can be found in http://code.google.com/p/frolog/source/browse/trunk/GameScenarios/FairyTaleCastle/SourceGrammarXMG/

2.2. The SOURCE lexicons

The lemma lexicon:

This is the entry for the lemma "open", notice the SEM field, it gives information for the binaryRel XMG class specified in previous page. It says that the theta1 interface variable takes the value agent when the n0Vn1 tree is anchored using the lemma open.

```
*ENTRY: open
*CAT: v
*SEM: binaryRel[theta1=agent,theta2=patient,rel=open]
*ACC: 1
*FAM: n0Vn1
*FILTERS: []
*EX: {}
*EQUATIONS:
*COANCHORS:
```

Using this information and the semantic macros generated by MetaTAG, lexconverter generates the lexicon for Geni.

The morphological lexicon:

This is the entry for open:

Inflexed form	root	morphological features
opened	open	[pos:v mode:ind tense:past]
opened	open	[pos:v mode:ppart]
opens	open	[pos:v mode:ind pers:3 num:sing tense:pres]
open	open	[pos:v mode:base]
open	open	[pos:v mode:imp pers:2 num:sing|plur tense:pres]
opening	open	[pos:v mode:ger]

Unfortunately, Geni morphological support is very restricted. Geni expects the morphological features as an input together with the semantics. That is, Geni is not using the morph info to rule out realizations such as "you opens the chest". For more information see http://websympa.loria.fr/wwsympa/arc/geni-users/2009-02/msg00000.html

NOTE: Geni documentation provides an specification of the inputs that Geni requires in http://trac.haskell.org/GenI/wiki/InputFormat. However, such specification is in Geni format and cannot be used for parsing. Such description is the description of the files depicted in yellow in Page 3 and samples of code are given in Section 2.4:

- geni grammar
- geni lemma lexicon
- geni semantic macros

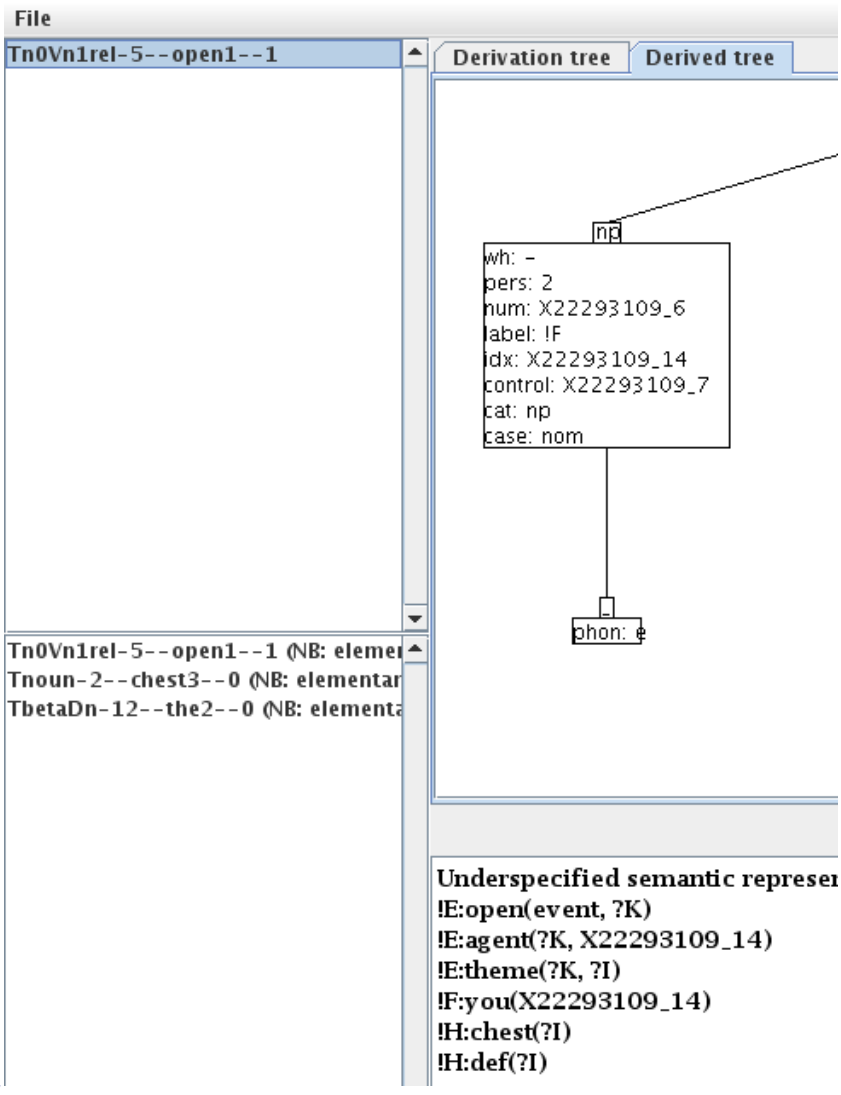
2.3. Using Tulipa to debug the grammar

If you want to create, debug or simply understand an XMG grammar there are two tools that are quite useful.

- The MetaTAG GUI showed in section 2.1
- Tulipa's GUI

Tulipa works in the opposite direction than Geni. From “open the chest” Tulipa produces `[!E:open(event, ?J) !E:agent(?J, X11508030_14) !E:theme(?J, ?L) !F:you(X11508030_14) !H:chest(?L) !H:def(?L)]` which is the semantics expected by Geni to realize “open the chest “

Here is an screenshot of tulipa when parsing “open the chest”. You can see:



- all the features of the nodes already instantiated.
- the derivation tree and the derived tree
- the tree fragments used with the uninstatiated features
- the resulting semantics
- the elementary trees that where not used because of feature unification failure.
- You can use Tulipa with GUI or you can make it write it's output to an XML file.
- The only drawback is that, if the grammar is big (let's say, more than 1000 TAG trees), Tulipa gets slow.
- For instructions on how to install the parser Tulipa see: http://sourcesup.cru.fr/tulipa/

2.4 (NEW) Geni object files (samples of the “files in yellow”)

Geni grammar

This one TAG tree for transitive verbs (and imperative subject) generated by geniconvert, as you see, you don't really want to modify it by hand ...

```
% ------------------------ Tn0Vn1rel-3
n0Vn1rel:Tn0Vn1rel-3 ( ! arg0:?T arg1:?O arg1L:?I1 arg2:?C1 arg2L:?D1 label0:?U label1:?P
objectI:?C1 objectL:?D1 rel:?F1 subjectI:?O subjectL:?I1 theta1:?G1 theta2:?H1 topL:?U vbI:?
T) initial
n0 [cat:s idx:?L inv:- label:?M]![assign-case:nom assign-comp:?A cat:s control:?B
extracted:- idx:?C inv:- label:?D mainv:?E mode:imp num:?F pass:?G perf:?H pers:2 progress:?
I tense:?J wh:-]{
 n1 aconstr:noadj [case:nom cat:np control:?B idx:?O label:?P num:?F pers:2 wh:-]![cat:np]{
  n2 type:lex "e" [phon:e]![phon:e]
 }
 n3 [assign-case:nom assign-comp:?A cat:vp idx:?C label:?D mainv:?E mode:imp num:sing|plur
pass:?G perf:?H pers:2 progress:?I tense:?J]![assign-case:?R assign-comp:?S cat:vp compar:-
idx:?T label:?U mainv:?V mode:imp num:?W pass:- pers:?X tense:?Y]{
  n4 type:anchor [assign-case:?R assign-comp:?S cat:v idx:?T label:?U mainv:?V mode:imp
num:?W pass:- pers:?X tense:?Y]![cat:v]
  n5 type:subst [case:acc cat:np idx:?C1 label:?D1 scopeL:?U wh:-]![cat:np]
 }
}
semantics:[?U:?F1(event ?T) ?U:?G1(?T ?O) ?U:?H1(?T ?C1) ?P:you(?O)]
trace:[SubjectAgreement ObjectSem PROSem anchorProj1 SubjectArgumentTop VerbCaseAssigner
VerbalTop TopLevelClass VerbSem ArgumentNodeSemantics SubjectArgument SubjectArg1Linking
ImperativeSubject ObjectArg2Linking VerbalArgument Subject anchorProj2 CanObject
VerbalNonInverted dian0Vn1active CanComplement NonRealisedSubject Dian0Vn1 active Object
binaryRel n0Vn1rel]
```

Semantic macros for Geni

This is the semantic macro for the binary rel semantic schema that you can see written in XMG in page 4.

```
binaryRel[rel=?G,theta1=?E,theta2=?B]
        semantics:[?A:?B(?C,?D) ?A:?E(?C,?F) ?A:?G(event,?C) ]
        interface:[arg0=?C,arg1=?F,arg2=?D,label0=?A,rel=?G,theta1=?E,theta2=?B]
```

Geni lemma lexicon

This is the entry for the lemma open:

```
open n0Vn1  %(?A_1 ?C_1 ?D_1 ?F_1 event)
equations:[interface.arg0:?C_1 interface.arg1:?F_1 interface.arg2:?D_1 interface.label0:?A_1
interface.rel:open interface.theta1:agent interface.theta2:patient]
filters:[family:n0Vn1rel ]
semantics:[?A_1:patient(?C_1 ?D_1) ?A_1:agent(?C_1 ?F_1) ?A_1:open(event ?C_1) ]
```

3. Using Geniserver

Frolog uses Geni in an interactive way. That means that Geni is started as a server once (see Section 3.1) and then Frolog send requests (that is, queries) to the server interactively (see Section 3.2).

3.1. Running geniserver

Frolog uses a binary of geniserver (compiled by Eric, because I never managed to compile geni on my own, geni compilation is well know for being difficult, Eric is being working on that and should be easier now).

The command line used by Frolog to start the server is:

```
geniserver
       -m GameScenarios/FairyTaleCastle/ObjectGrammarGeni/grammar.geni
       -l GameScenarios/FairyTaleCastle/ObjectGrammarGeni/lexicon.geni
       --rootfeat='[cat:s|np]'
```

- -m is the command line option for specifying the geni grammar.
- -l is the command line option for specifying the geni lemma lexicon.
- --rootfeat let you specify values for features at the root of the trees. These features are then mandatory, meaning that if the tree does not have those feature values in the root, the corresponding realization is discarded. Frolog only wants trees such that the category of the root is s or np.

For other options of the command line of geniserver see
http://trac.haskell.org/GenI/wiki/CommandLineOptions

(NEW) Note: The default for rootfeat is `[cat:s inv:- mode:ind|subj wh:-]`

3.2. Querying geniserver

Once that geniserver is running, clients can connect to it through sockets. Alexandre Denis implemented a java client for sending queries to Geni and parsing it's answers. The code is available here: http://code.google.com/p/frolog/source/browse/trunk/NLPModules/Realization/GeniSocket.java

So, from what we've seen so far in order to generate “open the chest” you should send the following semantics to geniserver (how? For instance, using Alexandre's client):

```
[E:open(event J) E:agent(J X) E:theme(J L) F:you(X) H:chest(L) H:def(L)]
```

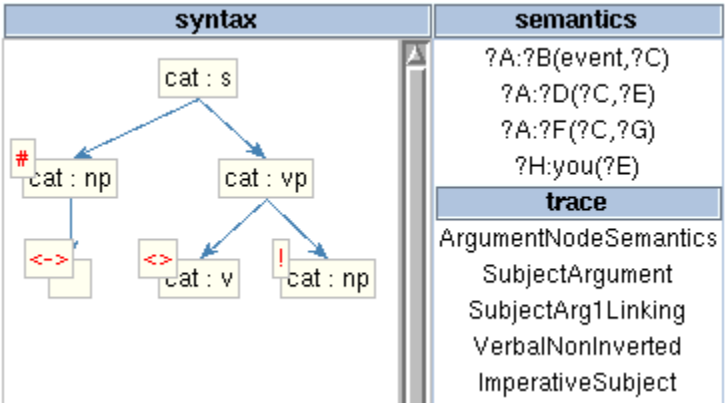
However, when we send this semantics to geniserver it outputs 2 things:

```
[open the chest, you open the chest]
```

Which makes sense given the grammar, there are two ways of obtaining this semantics (with Tulipa, for instance). Geni has a method for telling him which one you want. See next section.

3.3 Paraphrase selection

This is a screenshot of MetaTAG GUI repeated here:



The trace field lists the name of all XMG classes used to construct this TAG tree (in the screenshot the list is not complete).

If you want geni to use this TAG tree with ImperativeSubject instead of, say, the CanonicalSubject, you just need to tell Geni:

```
[E:open(event J)[ImperativeSubject] E:agent(J X) E:theme(J L) F:you(X) H:chest(L) H:def(L)]
```

You just associate to the semantic literal the name of the XMG class that you want Geni to use. See (Gardent and Kow, 2007) for theoretical details.

3.4 (NEW) Morphological selection

As we **discussed** during the talk I think that Geniserver accepts inputs like:

```
[E:open(event J) E:agent(J X) E:theme(J L) F:you(X) H:plural(L) H:chest(L) H:def(L)]
```

In order to produce outputs like “open the chests”. For this, the following line has to be added to the morphinfo file sent to geni when it starts.

```
plural    [num:pl]
```

For an example of a morphinfo file see the promettre example in
`geniroot/GenI/examples/promettre/morphinfo`

```
Notice that Geni command line accepts two morphological inputs that have different
information:
```

- `--morphinfo=FILE`
- `--morphlexicon FILE`

The morphlexicon file is in the format described in section 2.2.

4. Geni existing documentation

- Eric Kow, Geni developer (http://www.nltg.brighton.ac.uk/home/Eric.Kow/), who doesn't have much Geni time at the moment but does answer the mails about Geni.
- The old wiki (http://wiki.loria.fr/wiki/GenI/Getting_GenI)
- The new wiki (http://trac.haskell.org/GenI/wiki/), Eric encourages Geni users to use this wiki to document their findings.
- The mailing list, register and be part of the community! It's small but quite active lately :) (http://trac.haskell.org/GenI/wiki/MailingList)
- Literate geni, developer manual, not very user friendly (http://projects.haskell.org/GenI/literateGenI.pdf)
- Test suite (geniroot/GenI/examples)

References

L. Benotti. 2009. Frolog: An accommodating text-adventure game. In Proc. of EACL09. (demo)
B. Crabbe and D. Duchier. 2004. Metagrammar redux. In Proc. of CSLP04.
C. Gardent and E. Kow. 2007. A symbolic approach to near-deterministic surface realisation using tree adjoining grammar. In Proc. of ACL07.
C. Gardent. 2008. Integrating a unification-based semantics in a large scale lexicalised tree adjoininig grammar for french. In Proc. of COLING08.
L. Kallmeyer, T. Lichte, W. Maier, Y. Parmentier, J. Dellert, and K. Evang. 2008. TuLiPA: Towards a multi-formalism parsing environment for grammar engineering. In Proc. of the WGEAF08.