Integration of Attempto Controlled English (ACE) into Amine platform

By

Mohamed Nasri, 2013

Introduction

1. The knowledge representation formalism in Amine: CG, benefits, complications and limitations.

As cited here, Amine implements and adopts Conceptual Graphs (CG) as its main knowledge representation formalism. CG remains a powerful formalism for expressing knowledge and writing specifications, it expresses meaning in a form that is logically precise, humanly readable and computationally tractable. With their direct mapping to language, conceptual graphs serve as an intermediate language for translating computer-oriented formalisms to and from natural languages. With their graphic representation, they serve as a readable, but formal design and specification language (inspired from link).

CG remains however difficult and not obvious, especially for users not familiar with this formalism or while formulating complicated situations, thereby reducing the use of the platform to the CG experts.

2. Introduction to controlled natural languages and ACE

It would be interesting for users to avoid having to express knowledge using such specific formalism that requires a degree of expertise; it is more convenient for humans to use their natural language.

Given the numerous complexities and ambiguities of natural language, many researchers have chosen to focus on one subset of natural language, conventionally called "controlled natural language".

Controlled natural languages are subsets of natural languages, obtained by restricting the grammar and vocabulary in order to reduce or eliminate ambiguity and complexity.

The purpose behind the development and implementation of controlled natural languages is typically to aid non-native speakers of a natural language to learn it, speak it and understanding it, or to ease computer processing of a natural language. Thus, a controlled natural language can be used as high-level knowledge representation formalism to various kinds of knowledge based systems.

Developed by Attempto group, ACE is a controlled natural language that covers a well-defined subset of English that can be translated unambiguously into first-order logic via discourse representation structures and then be used for automated reasoning.

ACE is supported by various tools, among them the Attempto Parser Engine (APE) that translates ACE texts into a set of semantic elements called Discourse Representation Structure (DRS).

ACE supports questions and command sentences too and is able to treats modality sentences, like: admissibility, possibility, recommendation, provability and necessity.

Here are some examples of ACE sentences:

- A person is sick.
- The sick person takes the food carefully.
- Is John in the hospital?
- John does not take the medicine.
- Every person who coughs is sick.
- No customer is sick.
- The patient who takes the medicine heals.
- If the patient is sick then he should take the drug.

Here are the DRS structures for each of the above examples:

ACE text	**DRS structure**
A person is sick.	<DRS domain="A B C"> <object ref="A" noun="person" struct="countable" unit="na" numrel="eq" num="1" sentid="1" tokid="2"/> <property ref="B" adj="sick" degree="pos" sentid="1" tokid="4"/> <predicate ref="C" verb="be" subj="A" obj="B" sentid="1" tokid="3"/> </DRS>
The sick person takes the food carefully.	<DRS domain="A B C"> <property ref="C" adj="sick" degree="pos" sentid="1" tokid="2"/> <object ref="C" noun="person" struct="countable" unit="na" numrel="eq" num="1"

	sentid="1" tokid="3"/> <object ref="B" noun="food" struct="countable" unit="na" numrel="eq" num="1" sentid="1" tokid="6"/> <predicate ref="A" verb="take" subj="C" obj="B" sentid="1" tokid="4"/> <modifier_adv ref="A" adverb="carefully" degree="pos" sentid="1" tokid="7"/> </DRS>
If the patient is sick then he should take the drug.	<DRS domain=""> <Implication> <DRS domain="A B C"> <object ref="C" noun="patient" struct="countable" unit="na" numrel="eq" num="1" sentid="1" tokid="3"/> <property ref="A" adj="sick" degree="pos" sentid="1" tokid="5"/> <predicate ref="B" verb="be" subj="C" obj="A" sentid="1" tokid="4"/> </DRS> <DRS domain=""> <Recommendation> <DRS domain="D E"> <object ref="E" noun="drug" struct="countable" unit="na" numrel="eq" num="1" sentid="1" tokid="11"/> <predicate ref="D" verb="take" subj="C"

	obj="E" sentid="1" tokid="9"/> </DRS> </Recommendation> </DRS> </Implication> </DRS>

Table 1: ACE examples and their DRS structures.

3. Integration of ACE into Amine

1. ACE as another knowledge representation formalism, beside CG

As described above, ACE can be used as knowledge representation formalism, beside CG. This is important for users that are not familiar with CG formalism (or other formalism). Please note also that APE (ACE Parser) produces a DRS structure which is similar to CG as noted by J. Sowa: "Kamp's DRS notation is isomorphic to Peirce's existential graphs (EG), and conceptual graphs are a typed version of EGs".

Thus, our goal in integrating ACE in Amine is not to replace CG by ACE (or RDS), but to provide an alternative for those that are not familiar with Knowledge Representation formalisms (like CG).

2. ACE-CG mapping (mapping interface and ACE-CG GUI)

Since the Attempto Parser Engine (APE) translates the ACE text into a DRS structure, in which the items of the text are tagged with semantic information, we have developed and integrated into Amine a new module DRS2CG that uses the DRS structure in order to extract all the semantic elements and generate the corresponding conceptual graph. DRS2CG processes with all sentences form that can be processed with APE, particularly simple and composite ones.

Here are the two generated CGs corresponding to two ACE sentences extracted from the examples listed above:

ACE text	DRS structure	CG
A person is sick.	<DRS domain="A B C"> <object ref="A" noun="person" struct="countable" unit="na" numrel="eq" num="1" sentid="1"	[person]-propertyOf->[sick]

	tokid="2"/> <property ref="B" adj="sick" degree="pos" sentid="1" tokid="4"/> <predicate ref="C" verb="be" subj="A" obj="B" sentid="1" tokid="3"/> </DRS>	
A sick person takes the food carefully.	<DRS domain="A B C"> <property ref="C" adj="sick" degree="pos" sentid="1" tokid="2"/> <object ref="C" noun="person" struct="countable" unit="na" numrel="eq" num="1" sentid="1" tokid="3"/> <object ref="B" noun="food" struct="countable" unit="na" numrel="eq" num="1" sentid="1" tokid="6"/> <predicate ref="A" verb="take" subj="C" obj="B" sentid="1" tokid="4"/> <modifier_adv ref="A" adverb="carefully" degree="pos" sentid="1" tokid="7"/> </DRS>	[action_take : *p2] - -agentOf->[person:*p1]-propertyOf->[sick], -objOf->[food], -mannOf->[carefully]

Table 2: ACE examples with their DRS structures and the generated CGs.

With the integration of both DRS2CG and APE into Amine, this latter therefore has a module ACE2CG allowing to directly map ACE text to CG (The main class is aminePlatform.interfaces.ace.ace2cg.ACE2CG.java).

In order to facilitate the use of this module, a friendly graphical user interface (the ACE) has been created allowing the analysis of an ACE text and the generation of the corresponding CG (aminePlatform.guis.nlp.ace.Ace2CgGUI.java), the figures 2 and 3 print the analysis of the two examples bellow:

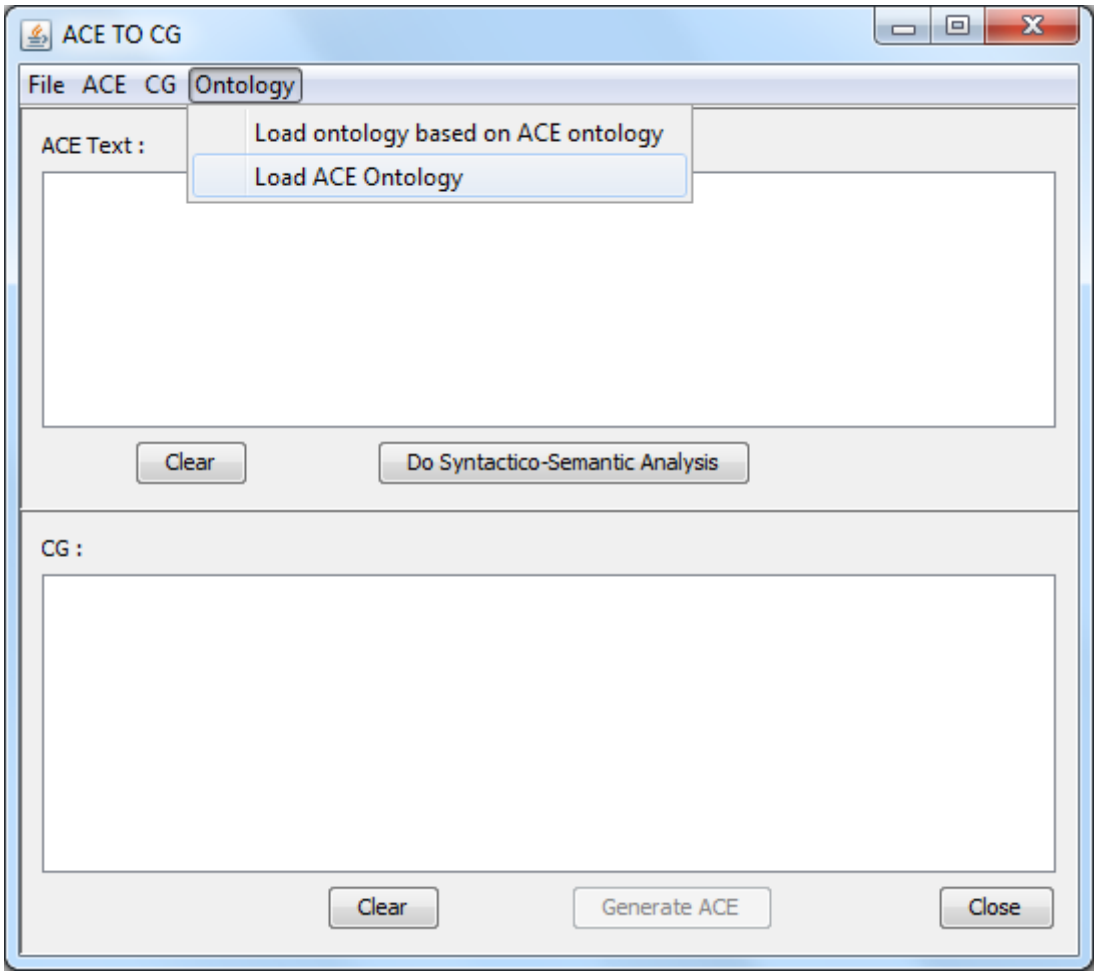


Figure 1: ACE2CG graphical interface, loading ACE ontology.

Step 1: User has to load ontology first (ACE ontology or another one, see the next session) and then do its analysis.
Step 2: User edit the ACE text and launch the analysis by activating the button “Do Syntactico-Semantic Analysis”.

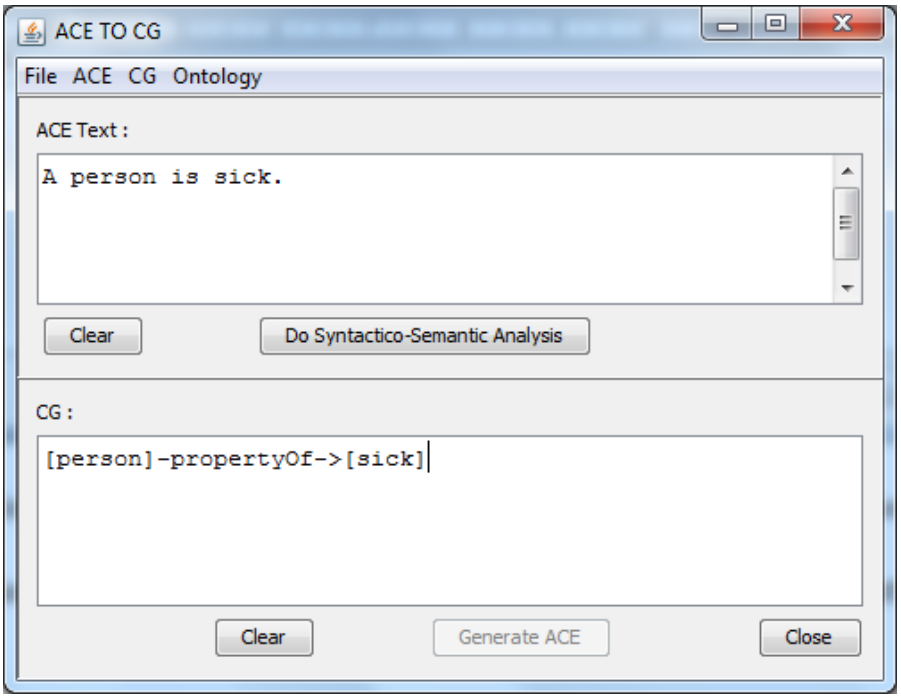


Figure 2: ACE2CG graphical interface, analysis of the sentence "A person is sick.".

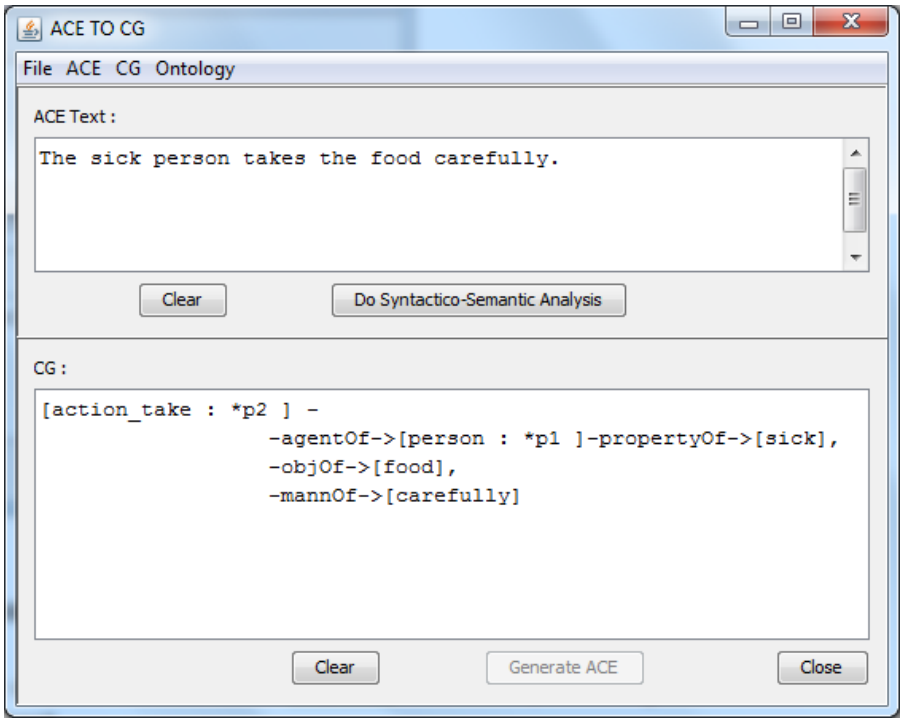


Figure 3: ACE2CG graphical interface, analysis of the sentence "The sick person takes the food carefully.".

3. ACE related ontology (the need for an ontology that supports ACE)

To recognize words by APE, words need to be defined in the ACE lexicon, otherwise, APE reports errors telling that the text or the sentence is incorrect.
In the same way that APE is based on the ACE lexicon, the DRS2CG mapping interface needs an ontology that contains all concepts found in the DRS structures, henceforth, this necessitates the creation of an ACE related ontology that should constitutes the ontological support for the ACE lexicon. The following figure illustrates this:

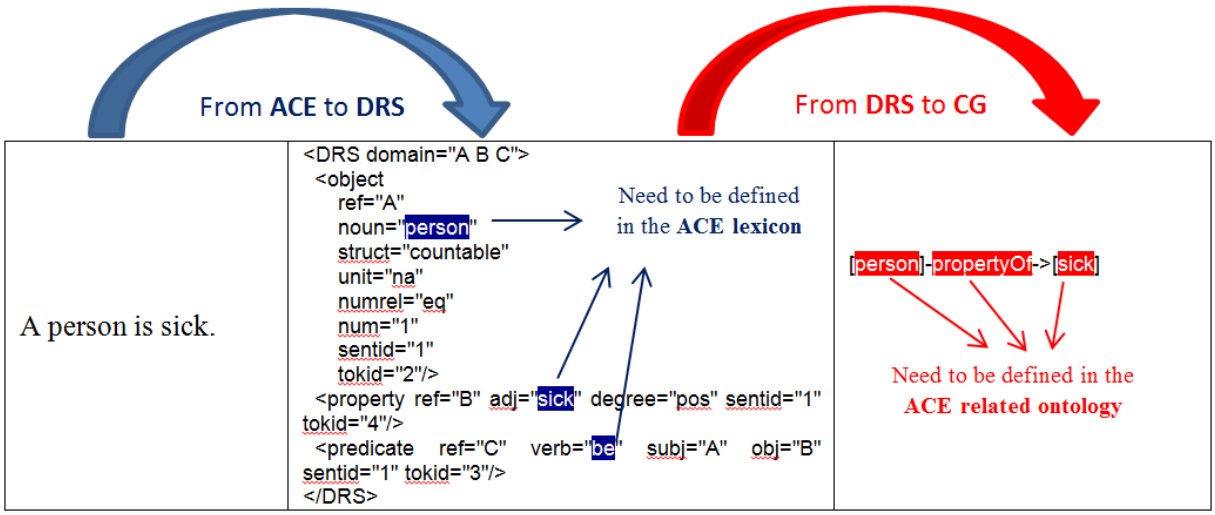


Figure 4: The usefulness of the ACE lexicon and the related ontology.

The ACE related ontology has been developed [Ref-a] by combining ACE lexicon and WordNet library [Ref-b]. For each ACE term, we get its lexical category from ACE (whether it is a verb, noun, etc. in order to place it under the action node, entity node, body_state node, etc.) and its super types (semantic information) via WordNet. Figure 15 shows a part of ACE lexicon and a part of the corresponding ontology.

```
noun_sg(person, person, human).
noun_sg(persona, persona, neutr).
noun_sg(personage, personage, human).
noun_sg(personal, personal, neutr).
noun_sg(personality, personality, neutr).
noun_sg(personation, personation, neutr).
noun_sg(personification, personification, neutr).
noun_sg(personnel, personnel, human).
```

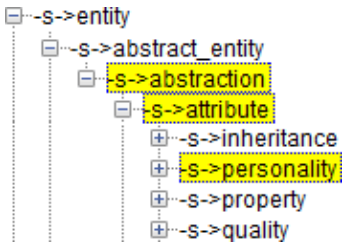


Figure 15. A part of the ACE related Ontology and the ACE lexicon

Some extra concept types (like cg_proposition, cg_ant, cg_conseq, cg_modality, md_possibility, md_necessity, md_admissibility, etc.) and relations (like agentOf, objOf, propertyOf, in, etc.) are added to the ontology because of their usefulness for the construction of the CG.
The development of the above mentioned ontology was not that straightforward. Indeed, statistics from the WordNet database allowed us to identify two categories of ACE terms:

- ACE terms that are covered by WordNet: this represents about 73% out of 52577 terms. Terms belonging to this category have been integrated with super types information extracted from WordNet.
- ACE terms that don't exist in WordNet content: fortunately, this is the smallest part of ACE terms (18% of the ACE terms). It is mainly made of compound terms like "act

for", "act in", "act like", "act out of", "act through", "act under" and "act with". Since these terms are basically a specialization of the type "act" which is part of WordNet synsets, they were considered as its subtypes. However, the picture is different for some other words. For instance, 'cornflake', 'decliner', 'deductibility' and 'evangelical' are not covered by WordNet and are not specializations of any WordNet synsets. Hence, these terms (the remaining 9%) have been directly integrated under "entity_root" node or "action" node depending on whether the term is a noun or a verb.

4. Using ACE instead of CG: extension of CG Notations GUI

Amine provides Tabbed panes for LF Editor, CGIF Editor and CG Graphic Editor that provide three editors for the corresponding CG notations. The user can select one of them to edit and/or to get an automatic translation of the current CG in a specific notation. If the used ontology is derived from the ACE ontology, a new tabbed pane is added to these three panes, to give user the new option to formulate knowledge in ACE too.

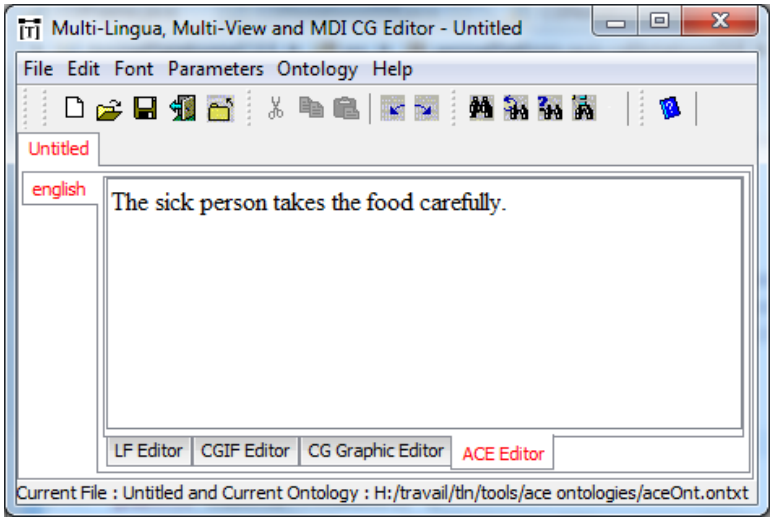


Figure 5.a: The ACE new editor.

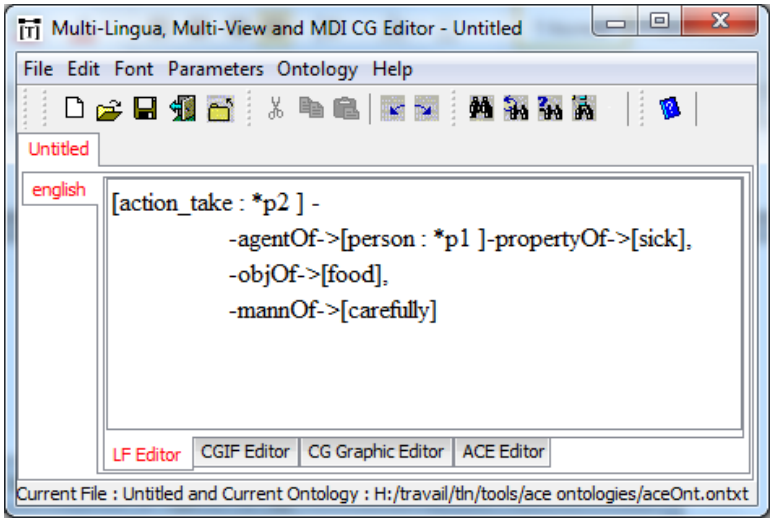


Figure 5.b: The LF editor.

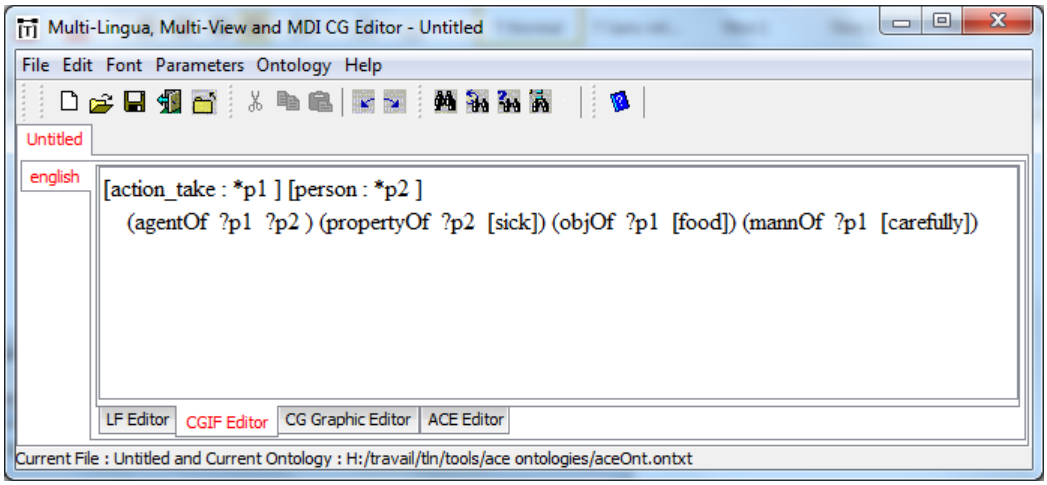


Figure 5.c: The CGLIF editor.

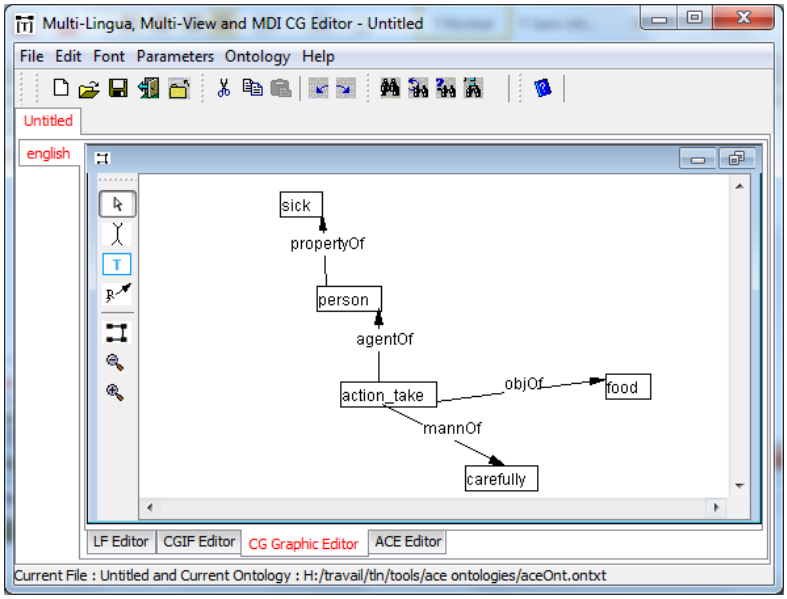


Figure 5.d: The CG Graphic Editor.

Please note that if the used ontology is not derived from ACE ontology, the "ACE Editor" panel is not provided.

5. ACE lexicon and ACE ontology can be extended

Even if ACE is a rich subset of the Standard English, it remains limited, so the possibility to enrich it (as well as the related ontology) is necessary.

To enrich ACE and its related ontology, two modes are possible: The first mode is to **Manually** edit the lexicon file and the related ontology, this way is not appreciated since a) the edition of the lexicon file requires a good expertise of its structure, and the required attributes for each word category (verbs, names, adjectives ...) and b) the extension cannot be done during the analysis, this latter has to be stopped and launched again once the extension succeeded. The second mode is via a **dynamic** extension, user is asked during the analysis via

custom graphical interfaces to inquire the required information for the word to add, once the addition succeeds, the analysis continues without being interrupted. This second mode has been implemented and integrated into Amine platform. Let's illustrate this by the extension of a noun, a verb and an adjective:

Example 1: Dynamic addition of a new noun

While analyzing the text: "A thirsty person drinks the coca.", APE does not recognize the word coca, Amine via an interactive interface reports this message to the user, asking him to check whether it's a spelling mistake (cocoa) or a really new word.

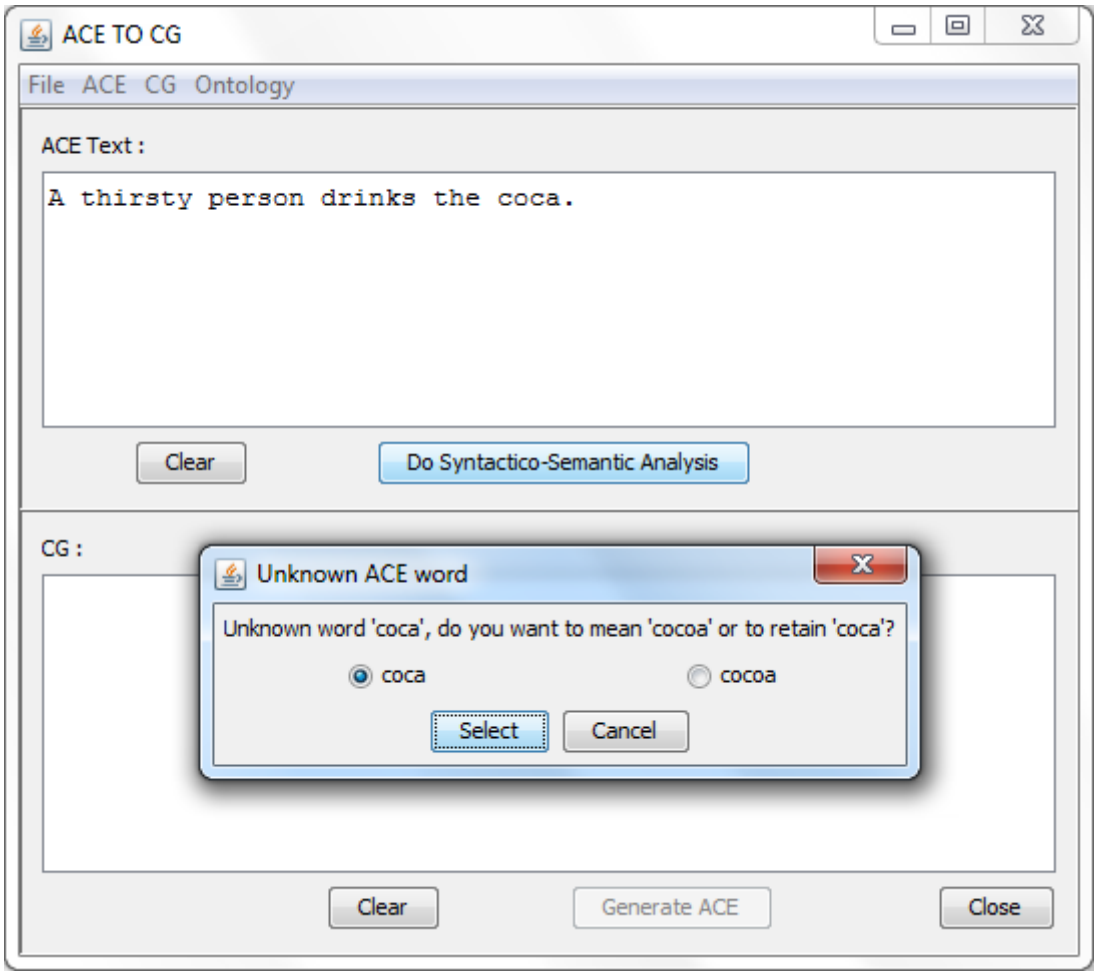


Figure 6.a: ACE2CG mapping, case of ACE unknown noun.

If it is a spelling mistake and the user means cocoa, then Amine replaces the word "coca" by "cocoa" and analyses the new text, the result of the analysis in this case becomes:

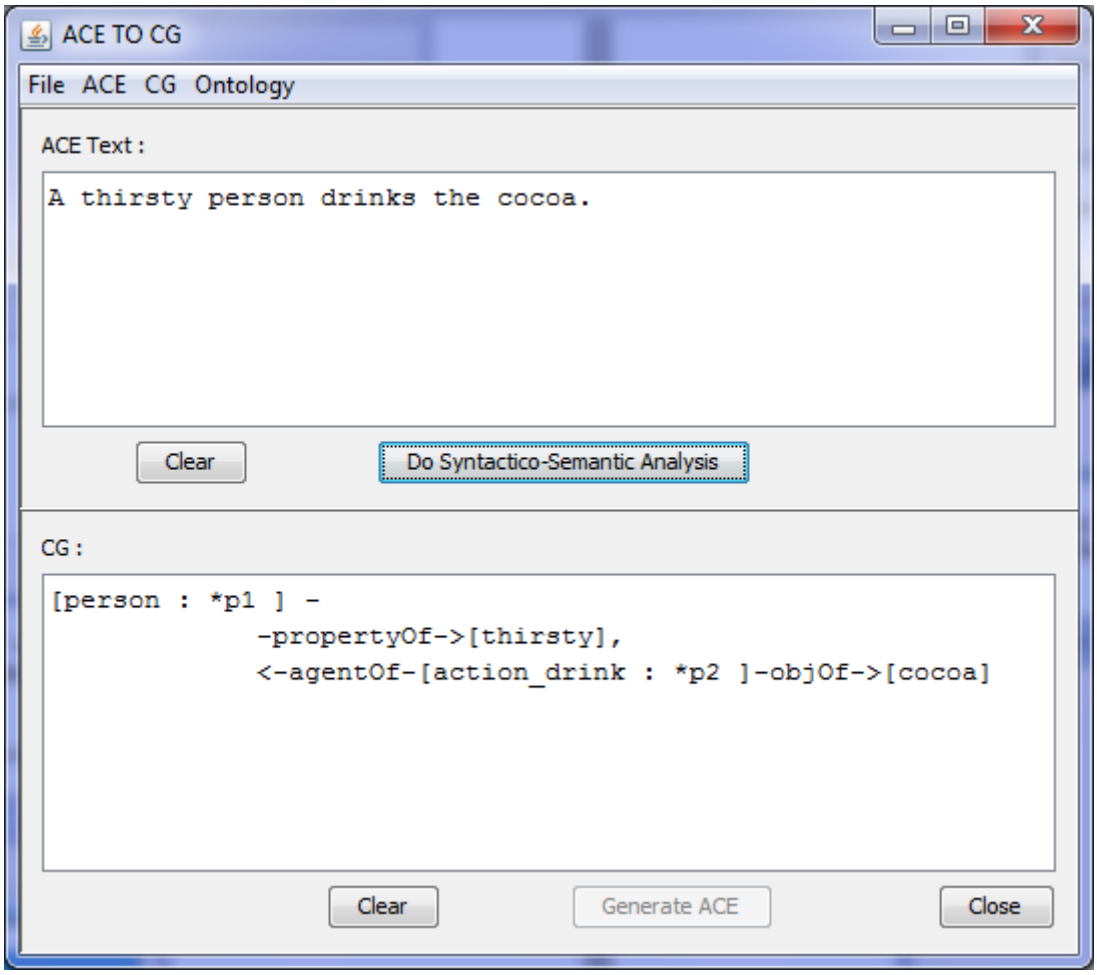


Figure 6.b: ACE2CG mapping, case of ACE spelling mistake.

Otherwise, if the user wants to retain the word "coca" which is not recognized by APE, then the addition process begins. This process is made of many steps depending on word family, let's continue with the same example:

Step 1) User has to select the term family, in our case, we select the "Noun" family:

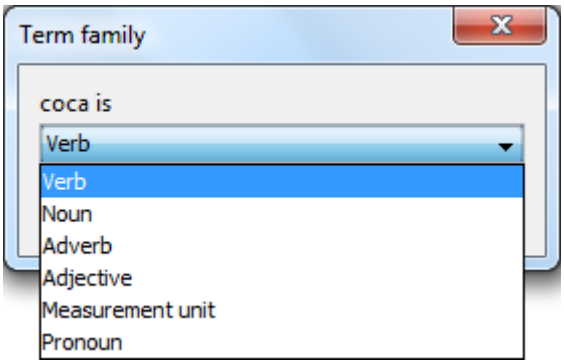


Figure 6.c: ACE2CG mapping, term extension: term family.

Step 2) Select the noun measurement type (countable, mass or both (dom))

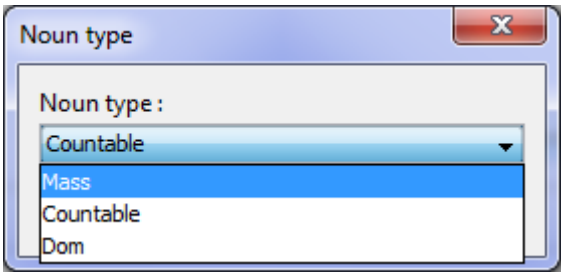


Figure 6.d: ACE2CG mapping, noun extension: noun type.

Step 3) Give the singular form of the noun.

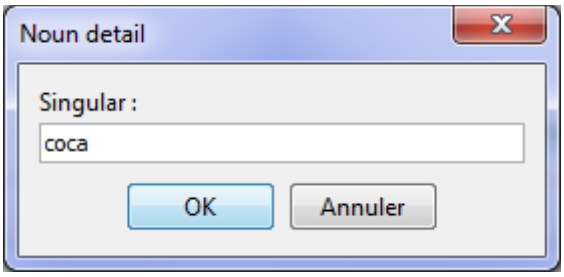


Figure 6.e: ACE2CG mapping, noun extension: noun detail.

Step 4) Specify whether the new noun is a human or not.

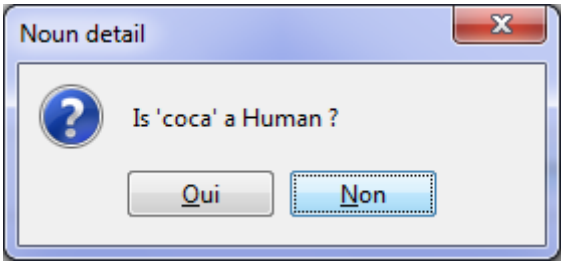


Figure 6.f: ACE2CG mapping, noun extension: is the noun used for humans?

Step 5) New type (corresponding to the new word) will be added to the ACE related ontology, user has to specify its direct super type and then activate the button “Save as subtype of ”.

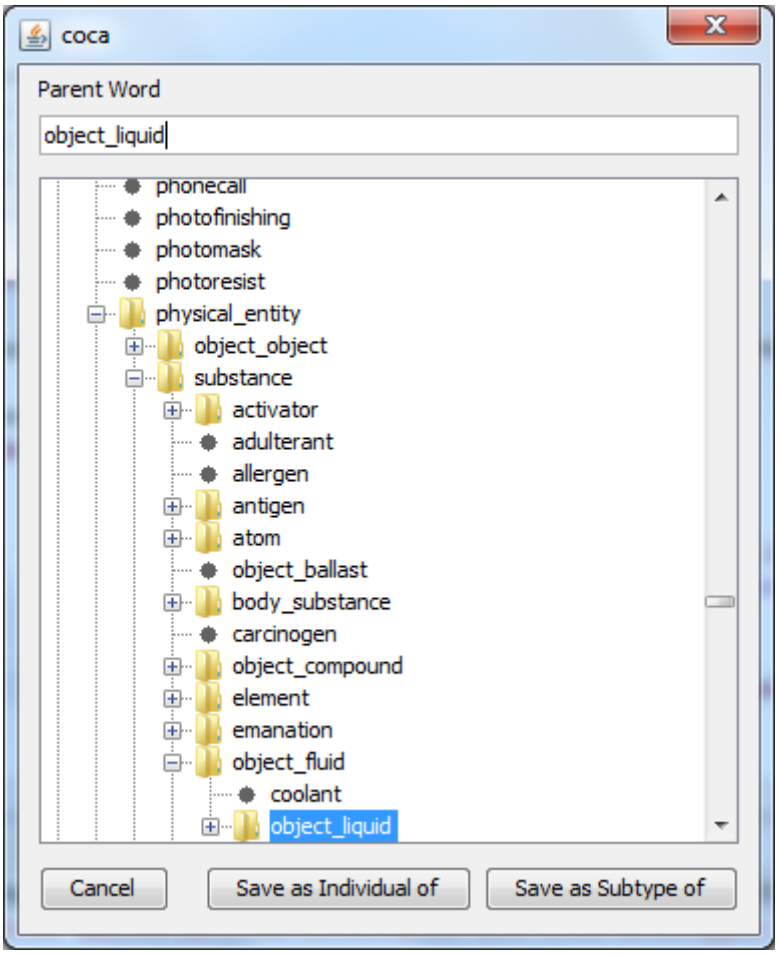


Figure 6.g: ACE2CG mapping, noun extension: extend the ontology

Once these information specified, Amine update the ACE lexicon, remake the APE program, add the new type to the ontology and launch the analysis again, figure below shows the new result of the analysis:

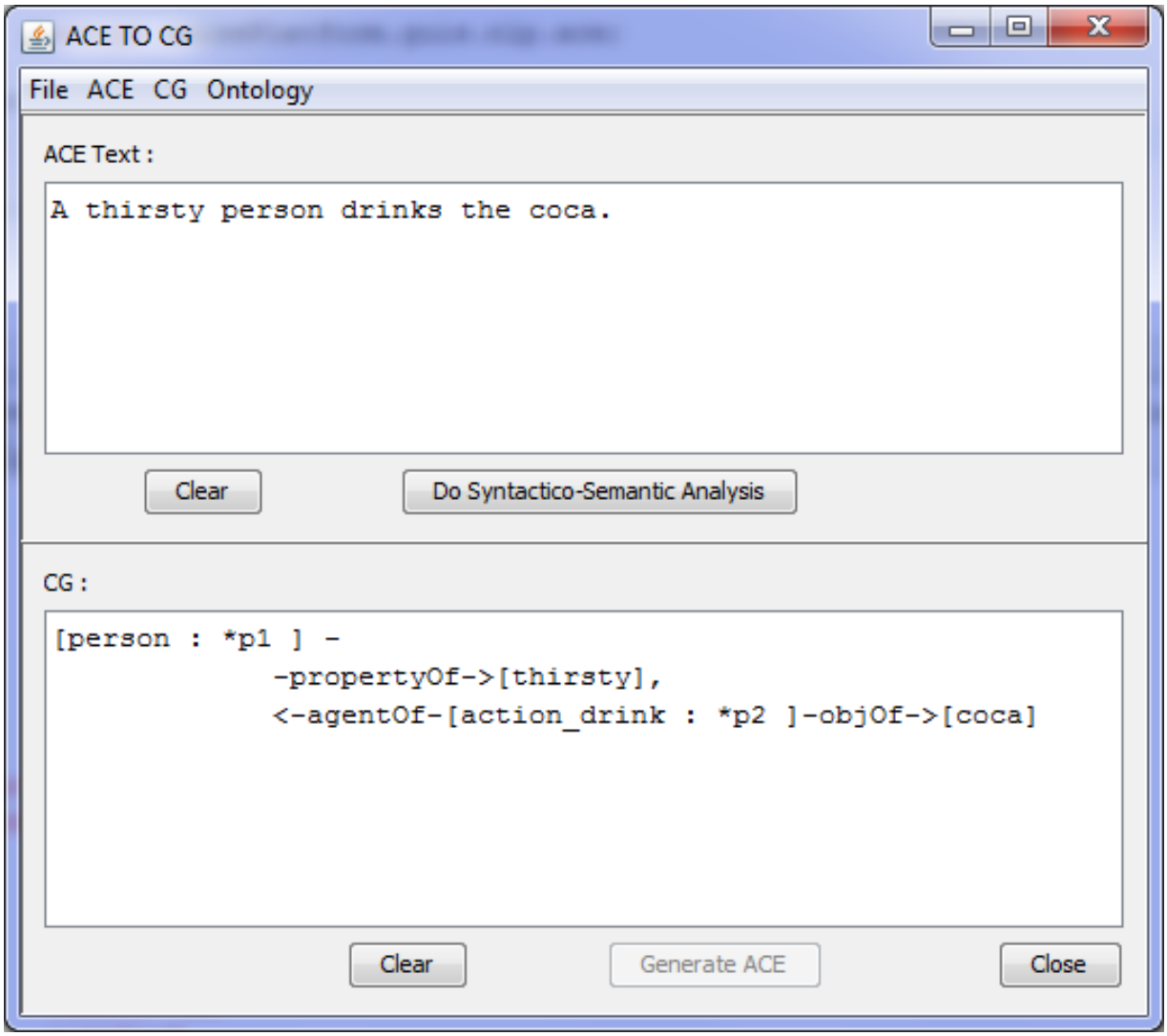


Figure 6.h: ACE text analysis after the extension of the new noun

Example 2: Dynamic addition of a new verb

Let's analyzing another example: "The boy blings-up the wardrobe.", APE does not recognize the verb "bling-up", so the process begins.

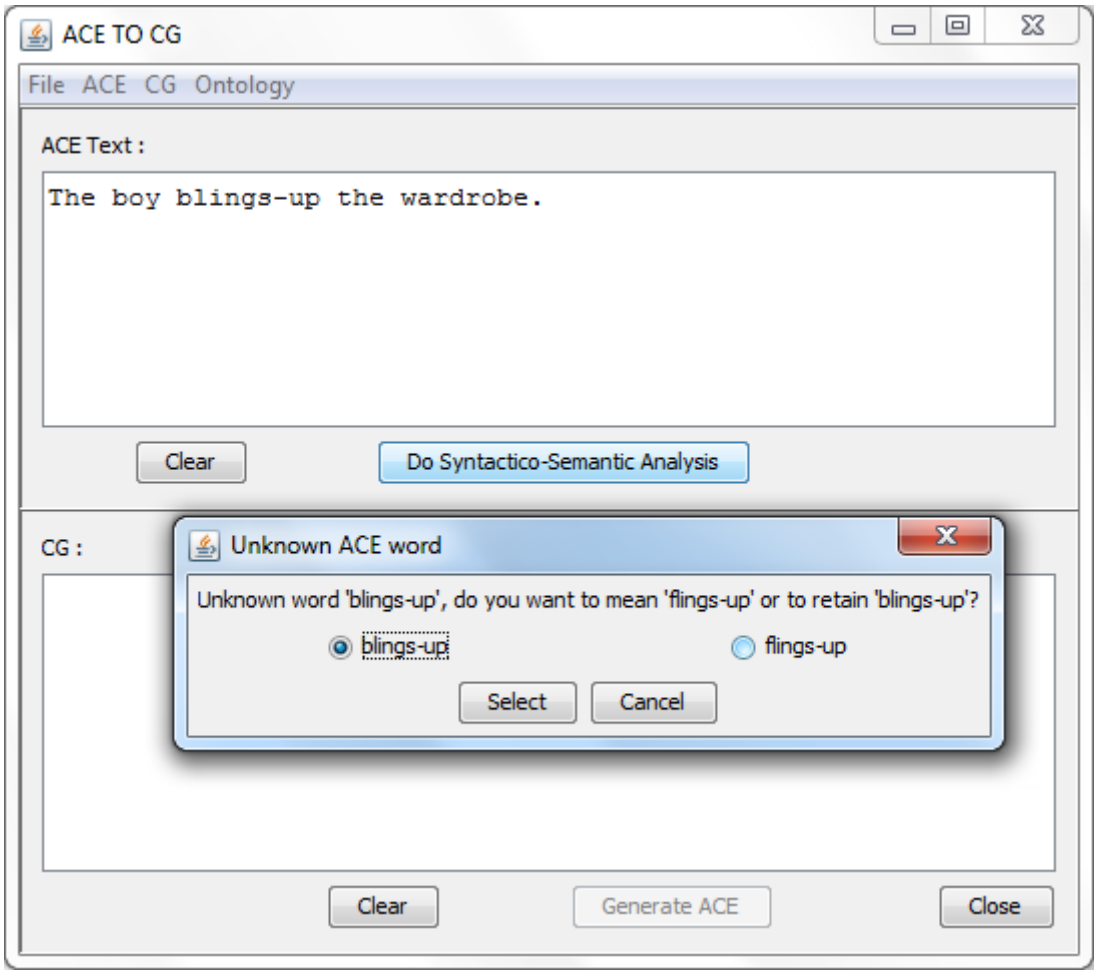


Figure 7.a: ACE2CG mapping, case of ACE unknown verb.

Step 1) User has to select the term family "Verb" family:

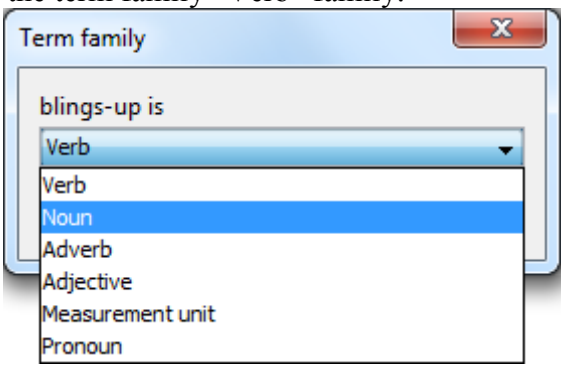


Figure 7.b: ACE2CG mapping, term extension: term family.

Step 2) Select the verb type (intransitive, transitive or ditransitive)

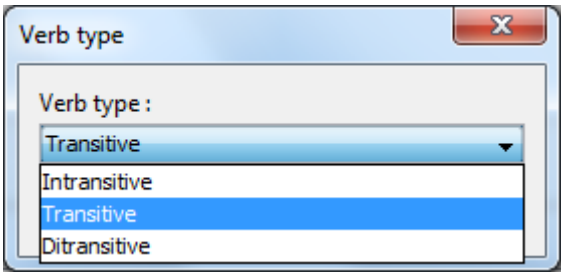


Figure 7.c: ACE2CG mapping, verb extension: verb type.

Step 3) 4) and 5) Give the verb in the infinitive mode and some details about its conjugation.

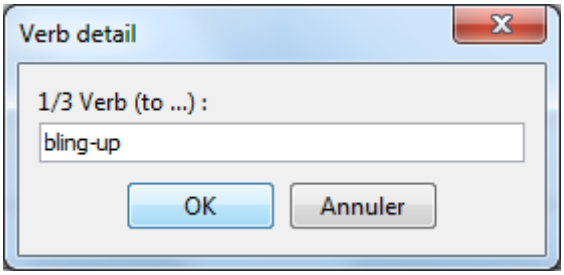


Figure 7.d: ACE2CG mapping, verb extension: verb detail.

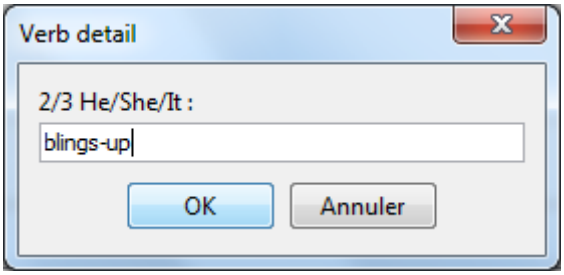


Figure 7.e: ACE2CG mapping, verb extension: verb conjugation 1.

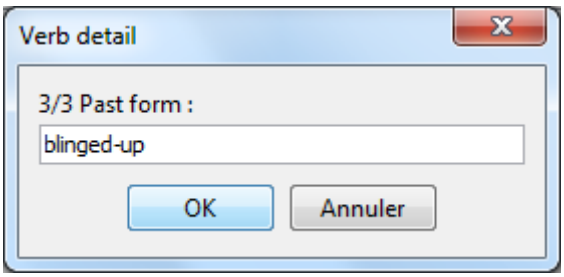


Figure 7.f: ACE2CG mapping, verb extension: verb conjugation 2.

Step 6) New action type (corresponding to the new verb) will be added to the ACE related ontology.

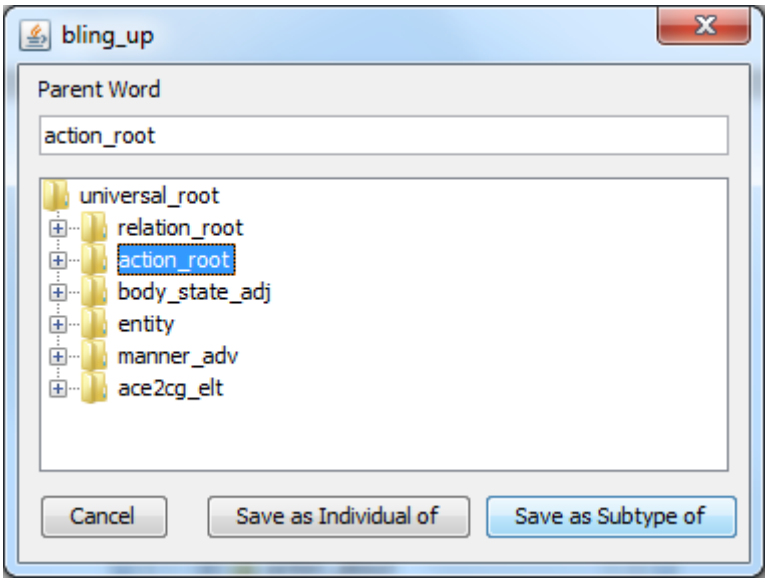


Figure 7.f: ACE2CG mapping, verb extension: extend the ontology

Once the verb is added, Amine launches the analysis again; figure below shows the new result of the analysis:

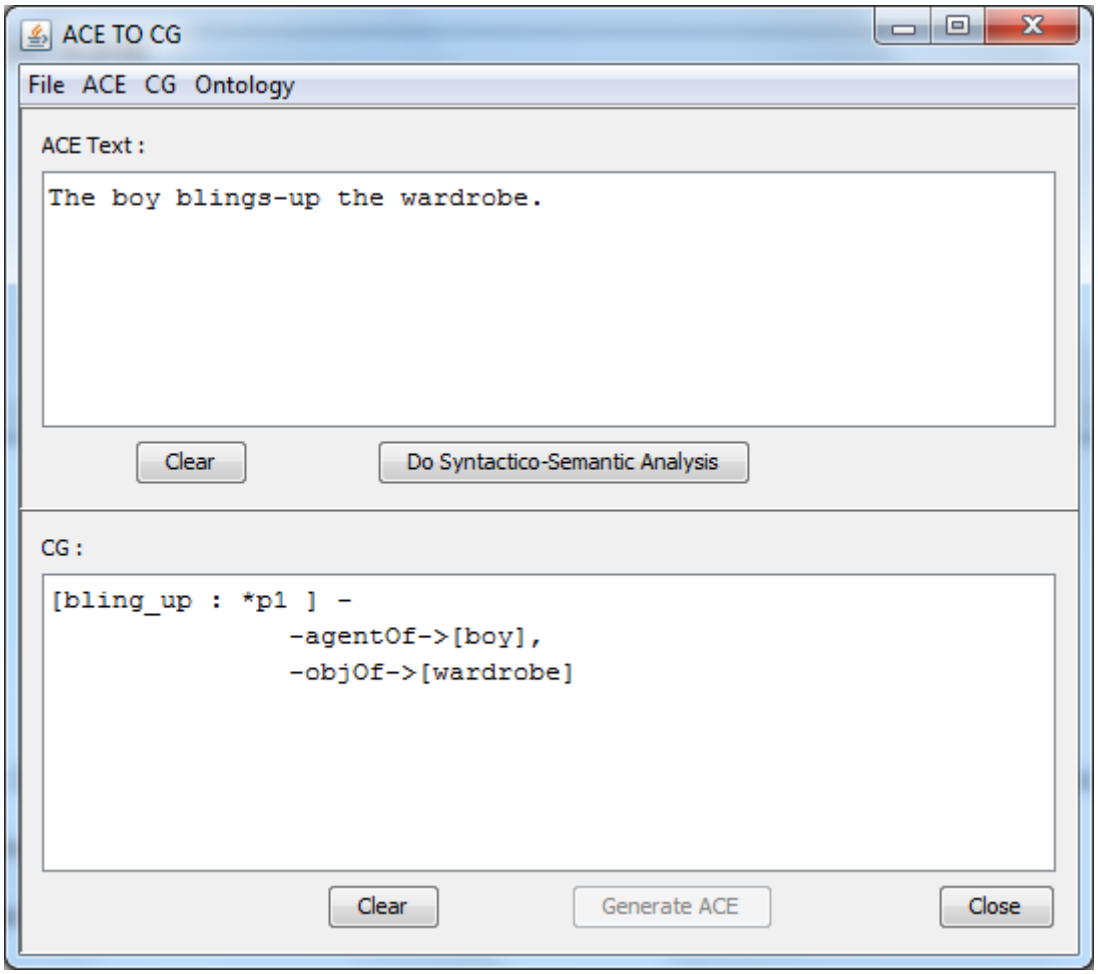


Figure 7.g: ACE text analysis after the extension of the new verb

Please note that compound words are hyphenated (connected using the hyphen "-") in ACE (example: stand-up, pick-up, etc.) and this is how they are stored in the lexicon. In ACEOntology, the hyphens are replaced by the underscore "_" (stand_up, pick_up, etc.), so the user is invited to use hyphen sign when using ACE and the underscore one when navigating or looking for supertypes in the ontology.

Example 3: Dynamic addition of a new adjective

As third example, let's analyzing the example: "The boy is superpowered.", APE does not recognize the adjective "superpowered", so the process begins.

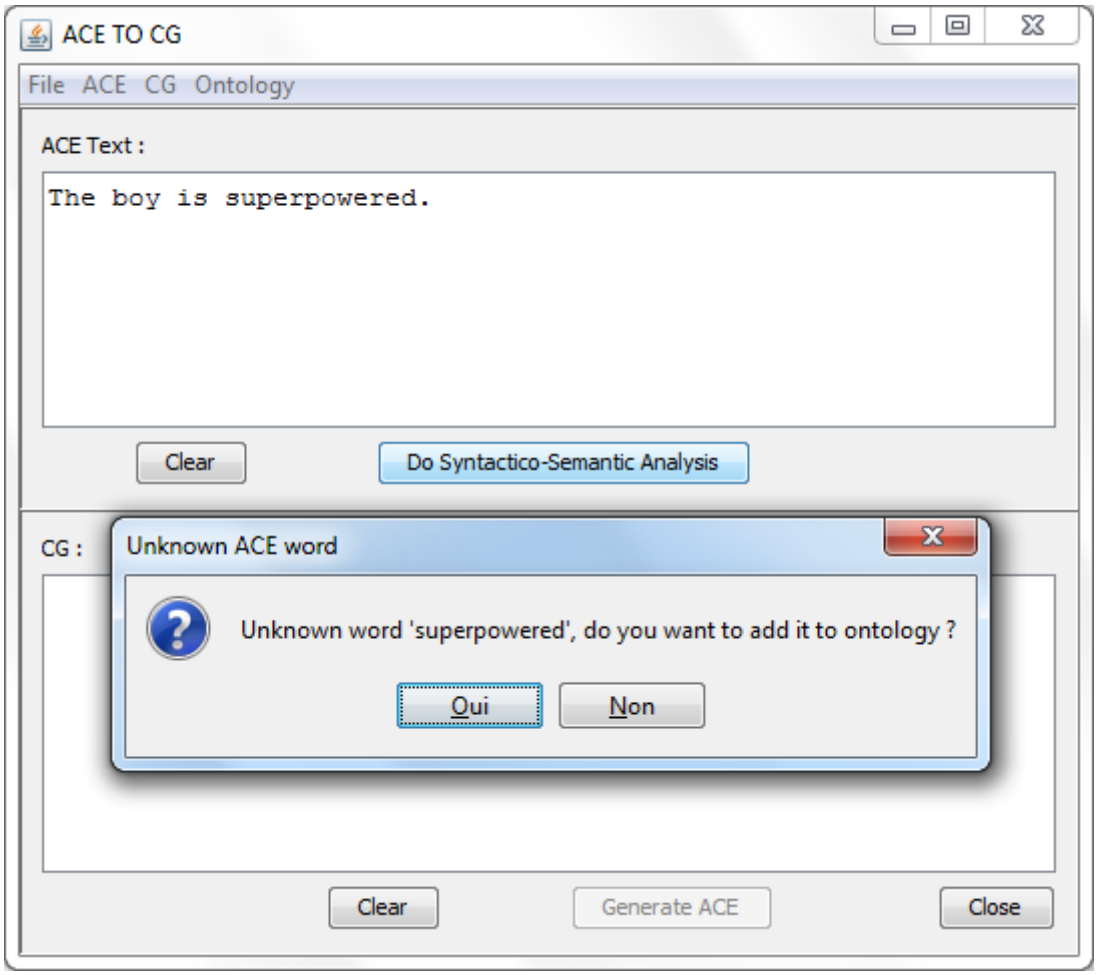


Figure 8.a: ACE2CG mapping, case of ACE unknown adjective.

Sep 1) User has to select the term family "Adjective" family:

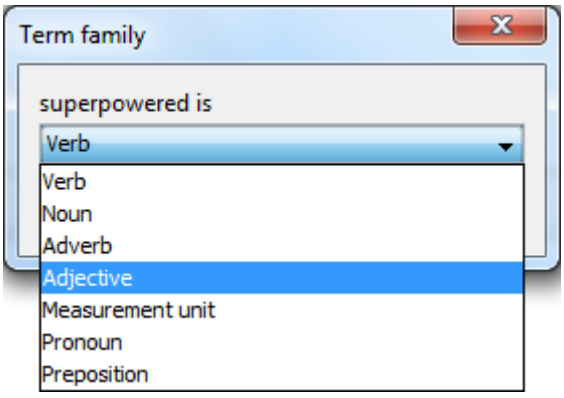


Figure 8.b: ACE2CG mapping, term extension: term family.

Step 2) Specify adjective form in comparison situations:

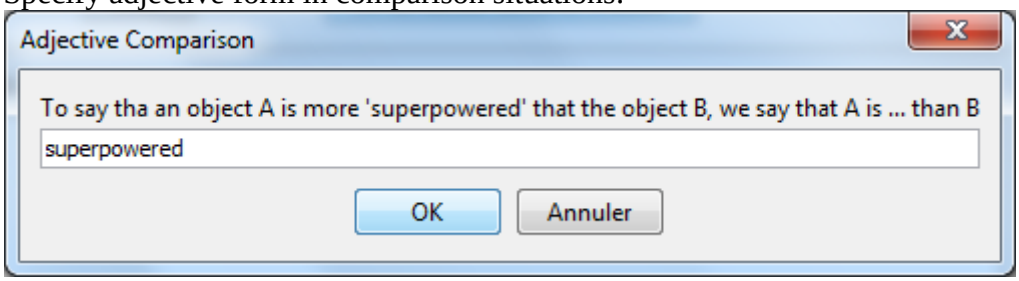


Figure 8.c: ACE2CG mapping, adjective extension: detail for comparison 1.

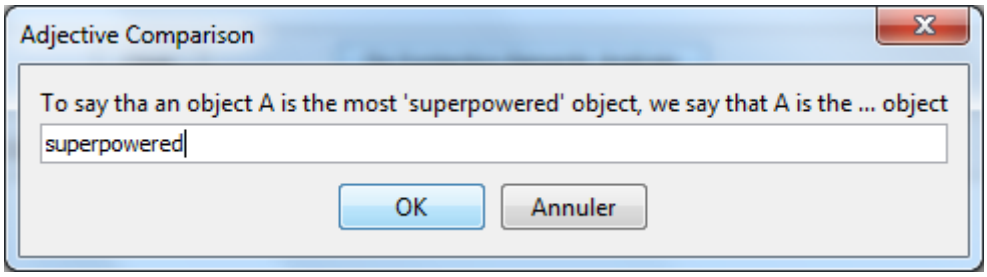


Figure 8.d: ACE2CG mapping, adjective extension: detail for comparison 2

Step 3) Tell whether the adjective is or not transitive:

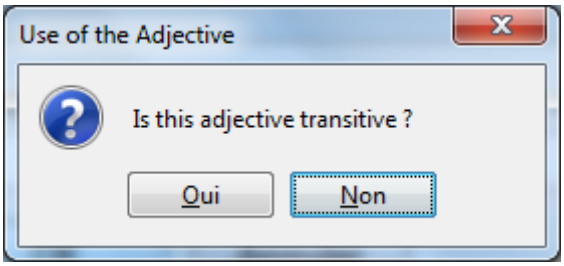


Figure 8.e: ACE2CG mapping, adjective extension: adjective transitivity.

Step 4) In the case of a transitive adjective, specify a preposition:

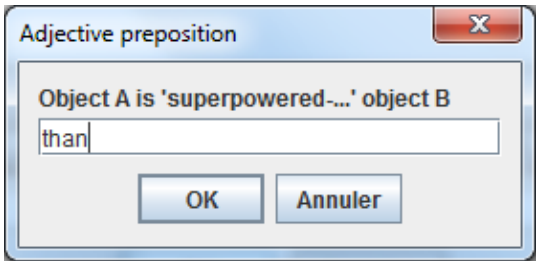


Figure 8.f: ACE2CG mapping, adjective extension: adjective preposition.

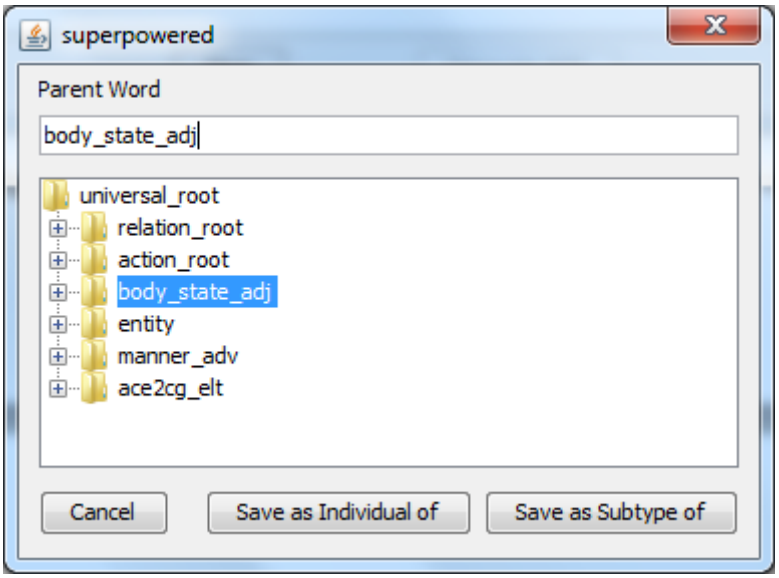


Figure 8.f: ACE2CG mapping, adjective extension: extend the ontology

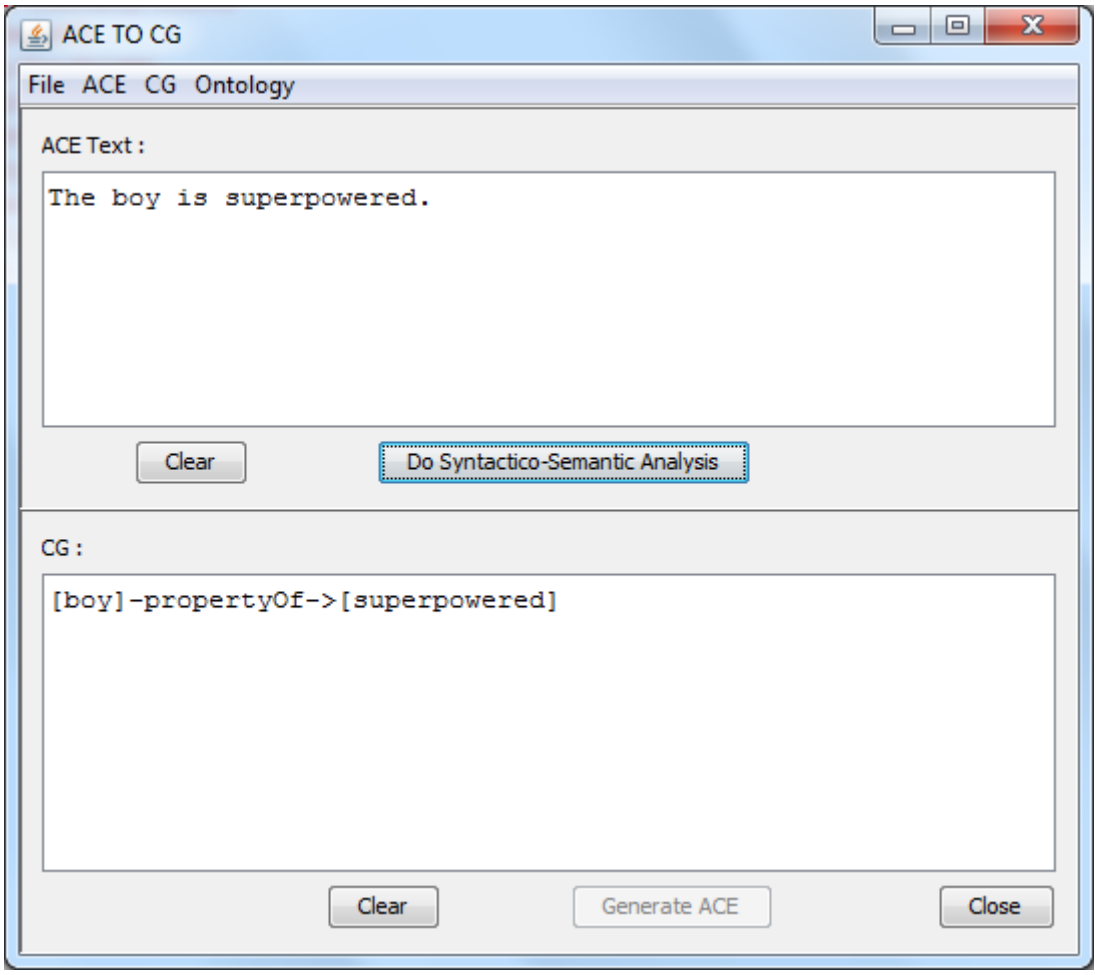
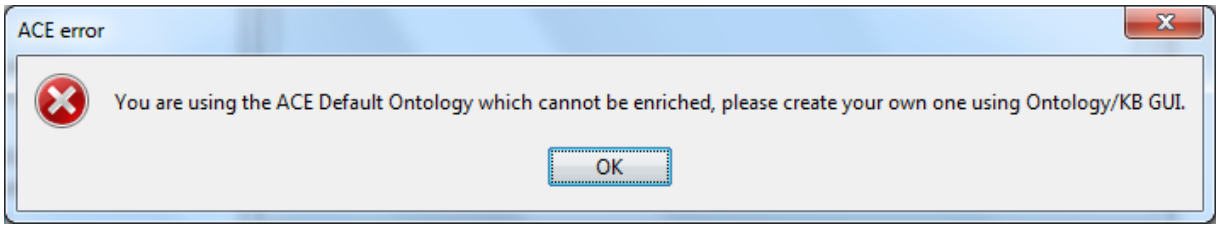


Figure 8.g: ACE text analysis after the extension of the new adjective

Note that the user can use the original/default ACE ontology/lexicon, but these two resources cannot be enriched, while attempting to enrich them Amine gives an error dialog:



Instead, the user can use his own ones or create new ones using the Ontology/KB GUI (as mentioned in the error message dialog). He/she can open the original/default ontology, using the Ontology/KB GUI, and then save it as a new one; a copy of the original/default ACE ontology as well as the associated lexicon will be created. Please save the new ontology with the suffix ".ontxt".

6. ACE for conceptual structures formulation

ACE has been integrated as an alternative wherever CGs are used, for instance, in the formulation of the conceptual structures (for the construction of ontology or knowledge base). User has then the options to formulate them using CG formalism or ACE controlled language, here are some illustrations of the use of ACE for this purpose:

Formulation of a type Definition using ACE: "The sugar is a sweet product."

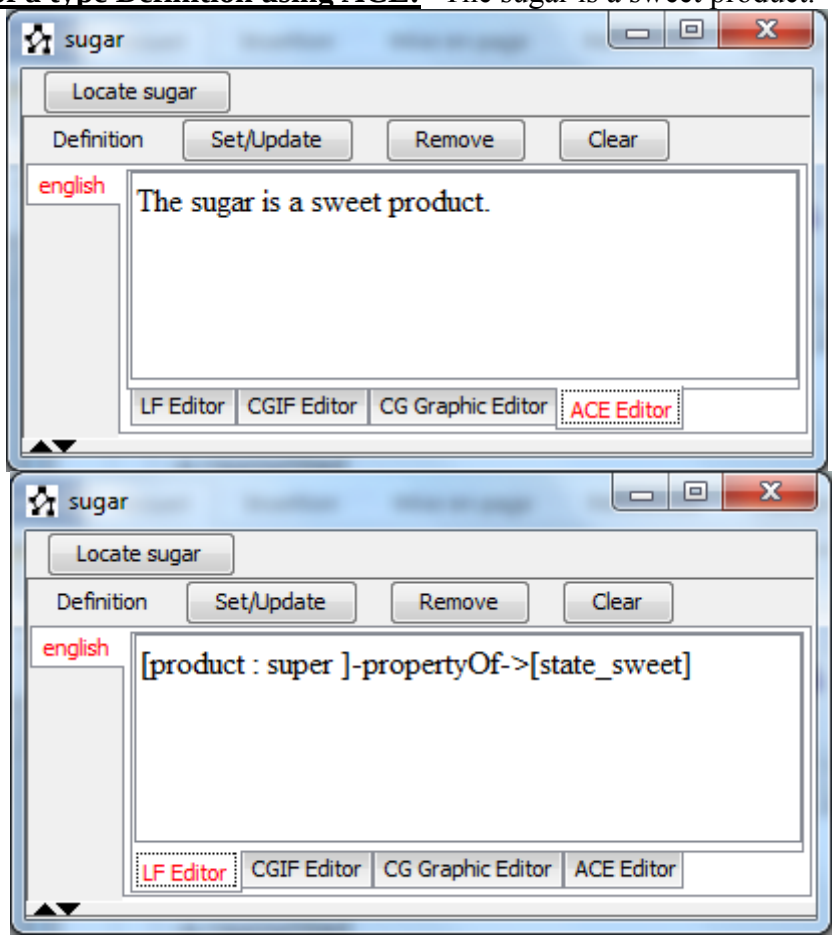


Figure 9.a: The definition of the type "sugar" in ACE and the generated CG in LF

Formulation of a Canon using ACE: "The living-thing drinks the liquid."

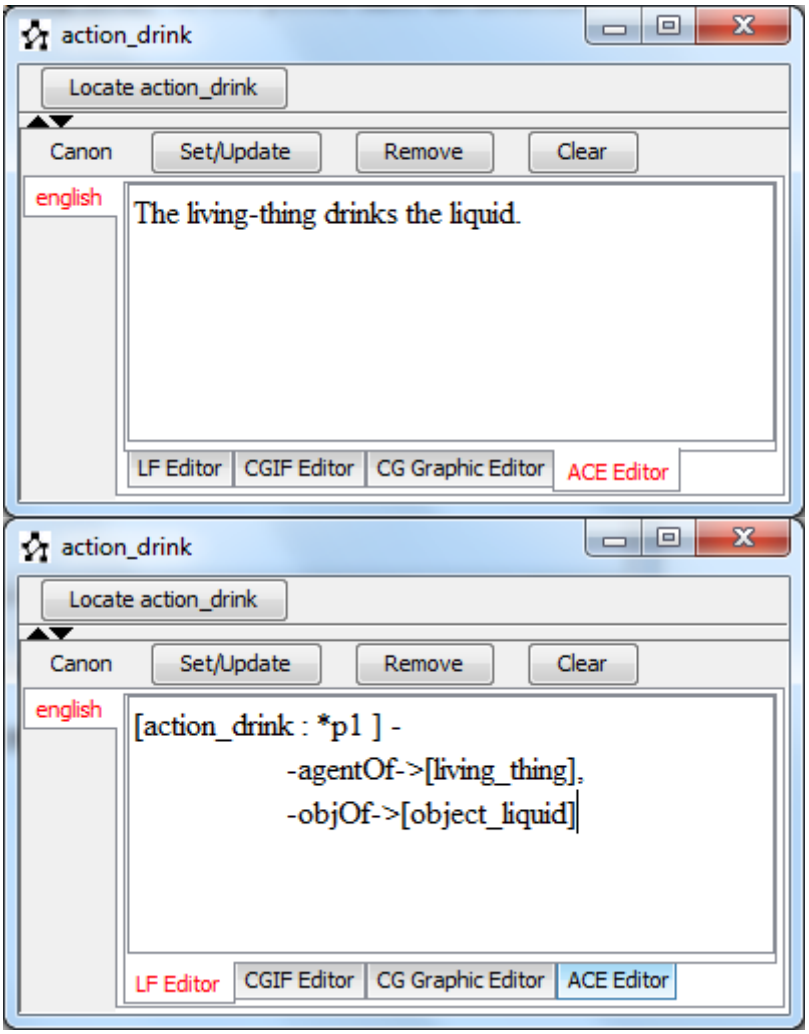


Figure 9.b: The canon of the action "drink" in ACE and the generated CG in LF

Formulation of a Situation using ACE: "A sick person takes the medicine."

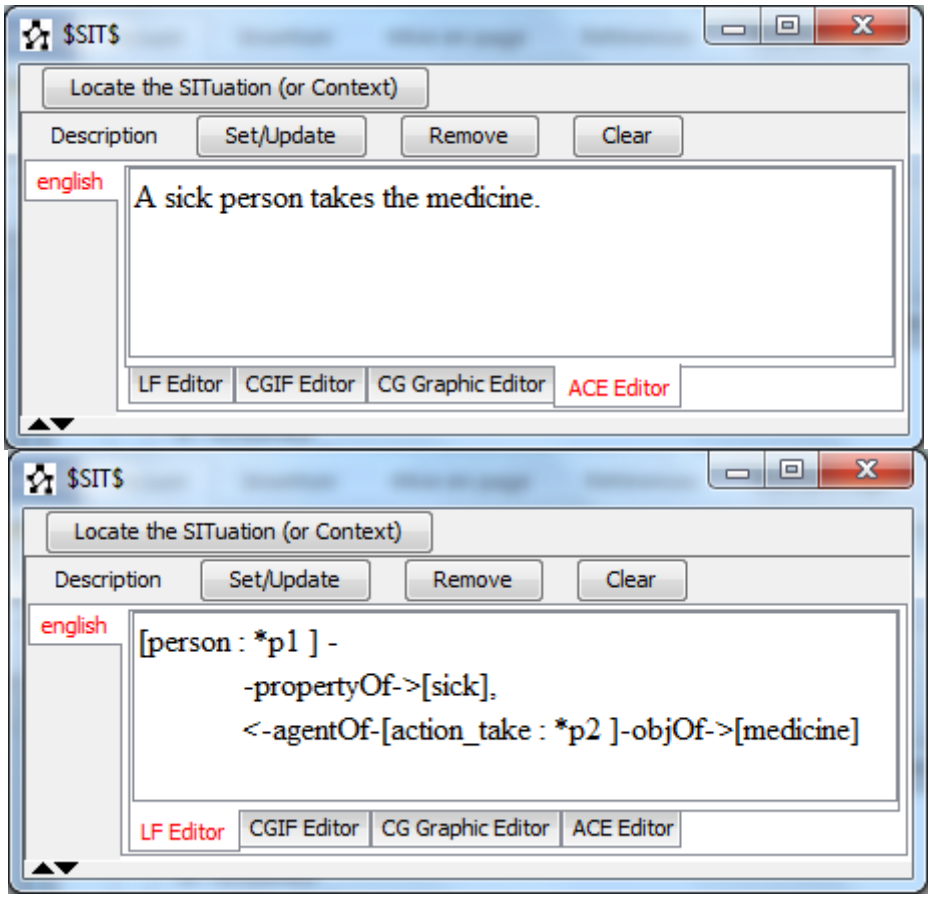
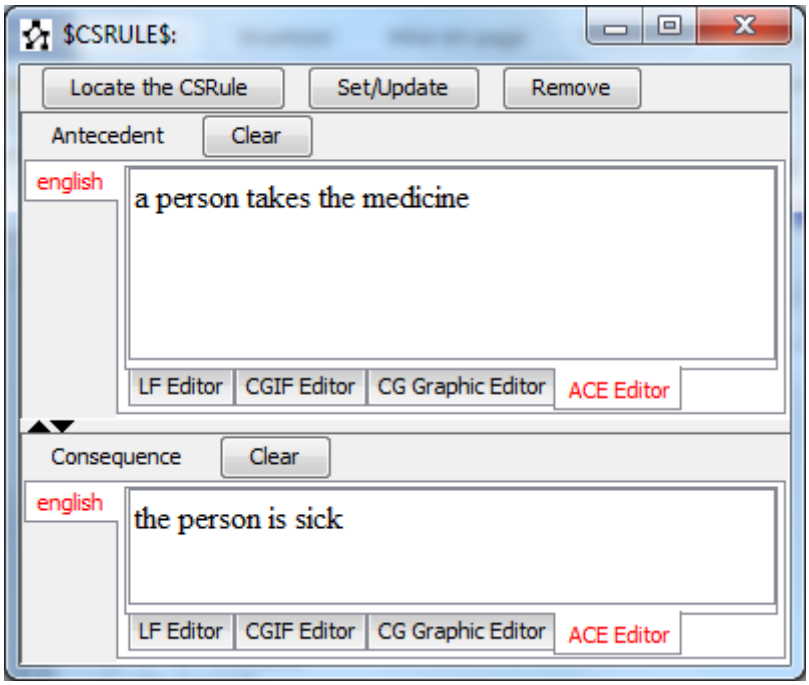


Figure 9.c: A situation of the type "sick" in ACE and the generated CG in LF

Formulation of a Rule using ACE: "If a person takes the medicine then the person is sick."



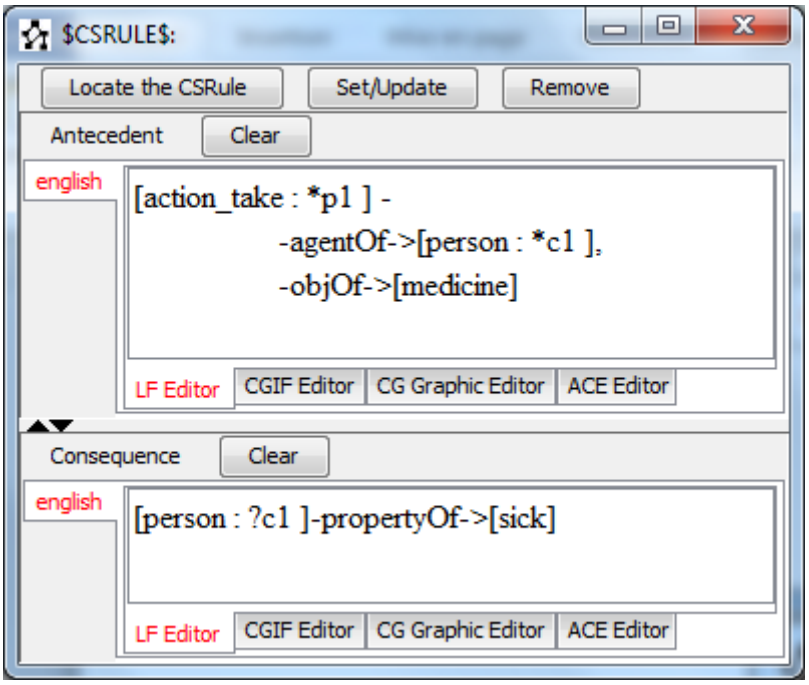


Figure 9.d: A rule of the type "sick" in ACE and the generated CG in LF

Please note that the formulation of these Conceptual Structures (CSs) using ACE is not complete; the user cannot formulate any CS using ACE. ACE does not provide a specific processing for these CSs. We have done some work to process canon, rule, situation, object definition, but action definition and relation definition are not possible for the moment.

7. ACE for ontology-based processes

If an ACE ontology is used, user can use ACE instead of CG to provide input for the ontology-based processes of Amine (like information retrieval, dynamic integration, explicitation, elaboration, inference, etc.) and for the other tools of Amine (like CGOperations GUI).

8. ACE for Information Retrieval and Question Answering (proof of concept)

With the integration of ACE and the CG-based operation (provided by Amine), Amine becomes more promising in the IR-QA fields.
To illustrate this point, we have developed a basic IR-QA system (for proof of concept only), this system analyses a text given by users (in ACE) and extracts its semantic. User can then ask questions about the meaning conveyed by the text. The system analyses the question and tries to extract the response.

For this purpose, the semantic of both the text and the question are extracted and are mapped to CGs, and via CG operation implemented by Amine, especially subsume (and subsumeWithResult) the system looks for the information in the text.

The figure below shows an example:

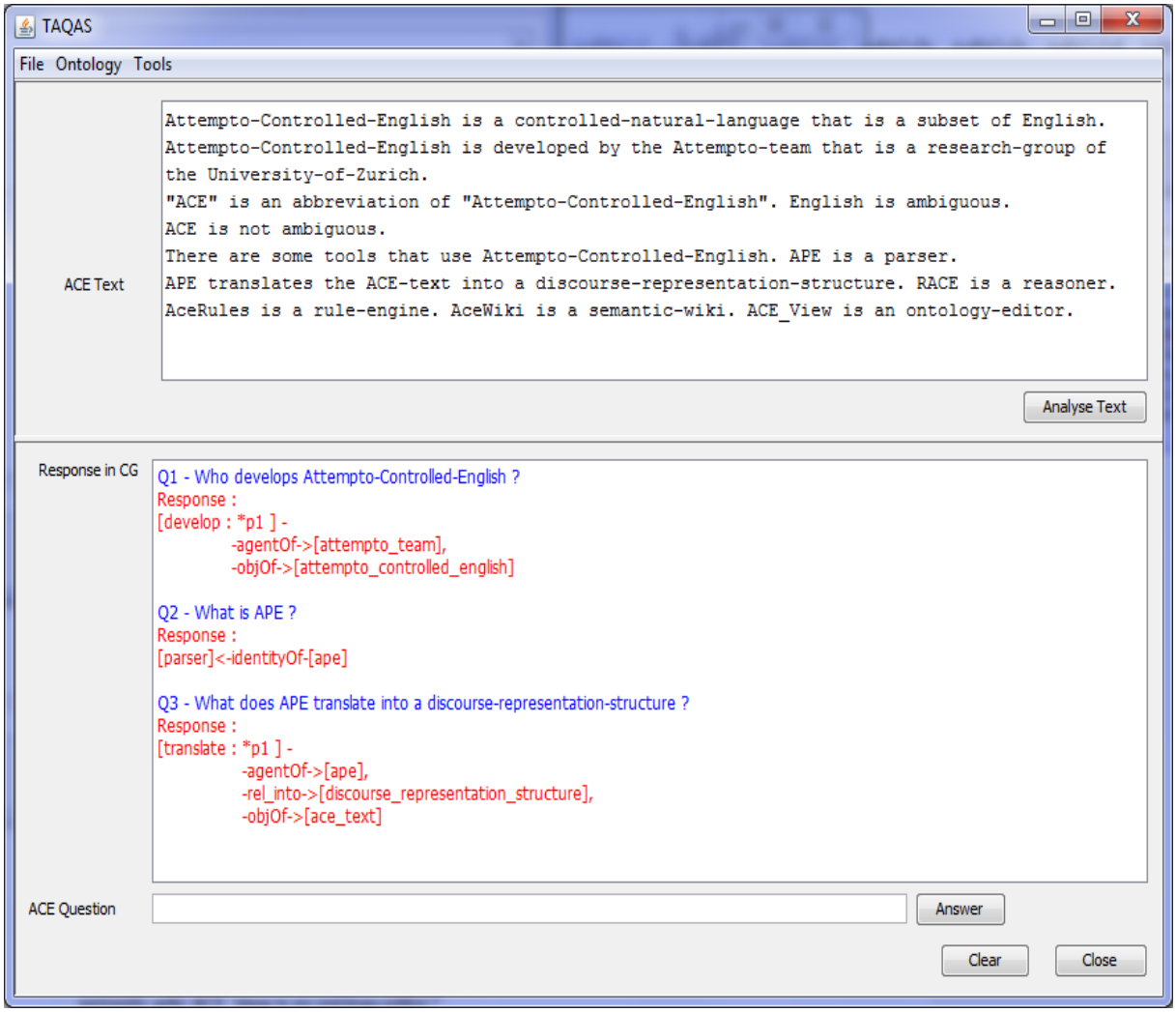


Figure 10: The IR-QA system

It should be noted that the purpose behind the development of this basic application was not a real IR-QA system, but only a proof of the concept. With ACE and CG operations, Amine becomes a good environment for the development of IR-QA systems.

9. Configuration issues

To use ACE in Amine, user has to download and install APE (which is an open source software that can be downloaded from this link) and to do some configurations in the "config.properties" file located in the root folder of Amine platform, the content of this file looks like:

```
#########################################################
#####    ACE PARSER ENGINE (APE) CONFIGURATION    #####
#########################################################

# APE path
ape_path=H:/tln/tools/ape/ape-6

#########################################################
#####          AMINE PLATEFORME CONFIGURATION      #####
#########################################################

# Amine ontology that supports ACE
ace_ont=H:/tln/tools/ace ontologies/aceOnt.ontxt
```

Figure 11: The configuration file

The first parameter (ape_path) is useful for the platform to launch the APE engine, it informs Amine about the absolute path of the APE folder while the second one (ace_ont) specifies the absolute path of the ACE related ontology file.

So please edit the "config.properties" file and fill these settings before launching Amine platform.

10. Runnable classes provided by ACE API:

aminePlatform.interfaces.ace.ace2cg.ACE2CG.java:

A class that loads the ACE related ontology and allows users to analyze ACE sentences.

```
10 juin 2013 18:39:25 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Loading ontology ...
10 juin 2013 18:39:57 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Ontology loaded in 31 seconds and 168 milliseconds.
ACE text :
John eats an apple.
10 juin 2013 18:40:09 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : John eats an apple. ...
[eat : *p1 ] -
          -agentOf->[masc :masc_john ],
          -objOf->[apple]
##########################################################################
ACE text :
If a person takes the medicine then the person is sick.
10 juin 2013 18:40:32 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : If a person takes the medicine then the person is sick. ...
[cg_ant : [action_take : *p1 ] -
                            -agentOf->[person : *c1 ],
                            -objOf->[medicine]
           ]-conseqOf->[cg_conseq : [person : ?c1 ]-propertyOf->[sick]
                          ]
##########################################################################
ACE text :
```

Figure 12: Example of an execution of the ACE2CG class using eclipse platform.

aminePlatform.applications.ace.Ace2CgGUI.java :

A friendly graphical interface developed to make the analysis easier for users. See figures 1, 2 and 3.

aminePlatform.samples.ace2cg.SamplesRunner

A class that runs the analysis of about 170 ACE sentences.

```
15 : John is rich.
12 oct. 2013 19:57:18 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : John is rich. ...
[masc :masc_john ]-propertyOf->[rich]
```

```
16 : John is a customer.
12 oct. 2013 19:57:18 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : John is a customer. ...
12 oct. 2013 19:57:19 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : John is a rich customer. ...
[masc :masc_john ]-identityOf->[customer]
```

```
17 : John is a rich customer.
[customer : *p1 ] -
               -propertyOf->[rich],
               <-identityOf-[masc :masc_john ]
```

```
18 : John is a rich and happy customer.
12 oct. 2013 19:57:19 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : John is a rich and happy customer. ...
12 oct. 2013 19:57:19 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : A customer has a valid card. ...
[customer : *p1 ] -
               -propertyOf->[rich],
               -propertyOf->[happy],
               <-identityOf-[masc :masc_john ]
```

Other examples:

John who is a clerk waits.

[pp_cg : [masc : *c1 masc_john]-identityOf->[clerk]

]-conjOf->[pp_cg : [masc : ?c1 masc_john]<-agentOf-[action_wait]

]

Every screen blinks.

[cg_ant : [object_screen : *c1]

]-conseqOf->[cg_conseq : [object_screen : ?c1]<-agentOf-[action_blink]

]

John is in the bank in the morning.

```
[masc :masc_john ] -
        -rel_in->[morning],
        -rel_in->[object_bank]
```

John who is a rich customer is in the bank in the morning.

```
[pp_cg : [customer : *p1 ] -
            -propertyOf->[rich],
            <-identityOf-[masc : *c1 masc_john ]
    ]-conjOf->[pp_cg : [masc : ?c1 masc_john ] -
                    -rel_in->[morning],
                    -rel_in->[object_bank]
            ]
```

Every man that owns a car or that owns a bike and that owns a house knows John.

```
[cg_ant : [pp_cg : [own : *p1 ] -
                -agentOf->[object_man : *c1 ],
                -objOf->[car]
            ]-or->[pp_cg : [pp_cg : [own : *p2 ] -
                            -agentOf->[object_man : ?c1 ],
                            -objOf->[object_bike]
                        ]-conjOf->[pp_cg : [own : *p3 ] -
                                        -agentOf->[object_man : ?c1 ],
                                        -objOf->[object_house]
                                    ]
                    ]
```

]-conseqOf->[cg_conseq : [action_know : *p4] -

-agentOf->[object_man : ?c1],

-objOf->[masc :masc_john]

]

The card of John is green.

[object_card : *p1] -

-propertyOf->[state_green],

<-possOf-[masc :masc_john]

A customer has a card that is valid.

[pp_cg : [object_card : *c1]-propertyOf->[valid]

]-conjOf->[pp_cg : [have : *p1] -

-agentOf->[customer],

-objOf->[object_card : ?c1]

]

A customer is more important than John.

[important : *p1] -

-opComp->[comp_than],

-targetOf->[masc :masc_john],

<-propertyOf-[customer]

John is fond-of Mary.

[fond_of : *p1] -

-targetOf->[fem :mary],

<-propertyOf-[masc :masc_john]

John is as fond-of Mary as of Sue.

```
[fond_of : *p1 ] -
        -opComp->[pos_as],
        -targetOf->[fem :mary ],
        <-propertyOf-[fem :fem_sue ],
        <-propertyOf-[masc :masc_john ]
```

A customer enters a card which is valid.

```
[pp_cg : [object_card : *c1 ]-propertyOf->[valid]
    ]-conjOf->[pp_cg : [enter : *p1 ] -
                -agentOf->[customer],
                -objOf->[object_card : ?c1 ]
            ]
```

A customer enters a card which is green and which is valid.

```
[pp_cg : *p1  [object_card : ?c1 ]-propertyOf->[valid]
        ] -
-conjOf->[pp_cg : [enter : *p2 ] -
                -agentOf->[customer],
                -objOf->[object_card : ?c1 ]
            ],
<-conjOf-[pp_cg : [object_card : *c1 ]-propertyOf->[state_green]
            ]
```

John waits or he inserts a card.

```
[pp_cg : [masc : *c1 masc_john ]<-agentOf-[action_wait]
    ]-or->[pp_cg : [action_insert : *p1 ] -
```

-agentOf->[masc : ?c1 masc_john],

-objOf->[object_card]

]

A customer believes that the card is correct.

[believe : *p1] -

-agentOf->[customer],

-objOf->[cg_prop : [object_card]-propertyOf->[state_correct]

]

A customer believes that the code is correct and the card is valid.

[pp_cg : [believe : *p1] -

-agentOf->[customer],

-objOf->[cg_prop : [object_code]-propertyOf->[state_correct]

]

]-conjOf->[pp_cg : [object_card]-propertyOf->[valid]

]

It is false that a screen blinks.

[cg_prop : [object_screen]<-agentOf-[action_blink]

]-propertyOf->[veracity : false]

John does not enter a card.

[cg_prop : [enter : *p1] -

-agentOf->[masc :masc_john],

-objOf->[object_card]

]-propertyOf->[veracity : false]

The card is not valid.

```
[cg_prop : [object_card]-propertyOf->[valid]
        ]-propertyOf->[veracity : false ]
```

John enters no code.

```
[cg_ant : [object_code : *c1 ]
        ]-conseqOf->[cg_conseq : [cg_prop : [enter : *p1 ] -
                                    -agentOf->[masc :masc_john ],
                                    -objOf->[object_code : ?c1 ]
                            ]-propertyOf->[veracity : false ]
                    ]
```

It is not possible that a customer enters a card.

```
[cg_prop : [cg_prop : [enter : *p1 ] -
                        -agentOf->[customer],
                        -objOf->[object_card]
                ]-propertyOf->[md_possibility]
        ]-propertyOf->[veracity : false ]
```

A customer must enter a card.

```
[cg_prop : [enter : *p1 ] -
                -agentOf->[customer],
                -objOf->[object_card]
        ]-propertyOf->[md_necessity]
```

It is necessary that a customer enters a card and a clerk waits.

```
[pp_cg : [cg_prop : [enter : *p1 ] -
                        -agentOf->[customer],
                        -objOf->[object_card]
                ]-propertyOf->[md_necessity]
        ]-conjOf->[pp_cg : [clerk]<-agentOf-[action_wait]
                ]
```

A customer shouldn't enter a card.

```
 [cg_prop : [cg_prop : [enter : *p1 ] -
                          -agentOf->[customer],
                          -objOf->[object_card]
                  ]-propertyOf->[md_recommendation]
          ]-propertyOf->[veracity : false ]
```

John's age is 80 years.

```
 [object_age : *p1 ] -
              -identityOf->[year : *p2 ]-countOf->[object_number : 80 ],
              <-attrOf-[masc :masc_john ]
```

A customer has less than 5 cards that are valid.

```
[pp_cg : [object_card : *c1 ] -
                    -countOf->[object_number : *p1 ]<-lessThan-[object_number : 5 ],
                    <-objOf-[have : *p2 ]-agentOf->[customer]
        ]-conjOf->[pp_cg : [object_card : ?c1 ]-propertyOf->[valid]
                ]
```

John inserts the card. The bank gives John the card.

```
[pp_cg : [action_insert : *p1 ] -
                    -agentOf->[masc : *c2 masc_john ],
                    -objOf->[object_card : *c1 ]
        ]-succOf->[pp_cg : [action_give : *p2 ] -
                            -agentOf->[object_bank],
                            -objOf->[object_card : ?c1 ],
                            -destOf->[masc : ?c2 masc_john ]
                ]
```

The customer is not happy and the code can be correct. It is not provable that the card is valid.

```
[pp_cg : [pp_cg : [cg_prop : [customer]-propertyOf->[happy]
                    ]-propertyOf->[veracity : false ]
            ]-conjOf->[pp_cg : [cg_prop : [object_code]-propertyOf->[state_correct]
                            ]-propertyOf->[md_possibility]
                    ]
        ]-succOf->[pp_cg : [cg_prop : [cg_prop : [object_card]-propertyOf->[valid]
                            ]-propertyOf->[md_provably]
                    ]-propertyOf->[veracity : false ]
                ]
```

If the code is valid then the machine accepts the card.

```
[cg_ant : [object_code]-propertyOf->[valid]
        ]-conseqOf->[cg_conseq : [accept : *p1 ] -
                            -agentOf->[object_machine],
                            -objOf->[object_card]
                    ]
```

If the code is valid then the machine accepts the card and John is happy.

```
[cg_ant : [object_code]-propertyOf->[valid]
        ]-conseqOf->[cg_conseq : [pp_cg : [accept : *p1 ] -
                                    -agentOf->[object_machine],
                                    -objOf->[object_card]
                                ]-conjOf->[pp_cg : [masc :masc_john ]-propertyOf->[happy]
                                    ]
                    ]
```

Is the card valid?

```
[cg_prop : [object_card]-propertyOf->[valid]
         ]-propertyOf->[cg_question : yes_no ]
```

Does John enter a card?

```
[cg_prop : [enter : *p1 ] -
                -agentOf->[masc :masc_john ],
                -objOf->[object_card]
         ]-propertyOf->[cg_question : yes_no ]
```

Where does John insert the card?

```
[cg_prop : [action_insert : *p1 ] -
                    -agentOf->[masc :masc_john ],
                    -objOf->[object_card],
                    -locOf->[location : V1 ]
         ]-propertyOf->[cg_question : where ]
```

Who enters what?

[cg_prop : [enter : *p1] -

-agentOf->[universal_root : V6],

-objOf->[universal_root : V7]

]-propertyOf->[cg_question : {what, who}]

Please run and consult the output of the class ‘aminePlatform.samples.ace2cg.SamplesRunner to get a more complete sample of sentences analyzed by ACE-Amine.

aminePlatform.applications.ace.taqas.TaqasGUI.java :

A graphical class for the previous TAQAS class. See figure 10.

aminePlatform.applications.ace.csAce2cg.aceTypeDef2cg. ACETypeDef2CG

A class that maps the type definitions formulated in ACE to CG, here is an example run under eclipse platform:

```
12 oct. 2013 19:49:49 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Loading ontology ...
12 oct. 2013 19:50:18 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Ontology loaded in 29 seconds and 97 milliseconds.
Example of definition :
-------------------------------------------
Word to define : Butcher
Definition : is a person who sells the meat.


Word to define :
rich
Definition :
a person who has the money
The definition to process : Rich is a person who has the money.
12 oct. 2013 19:50:54 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : Rich is a person who has the money. ...
Definition CG :
[have : *p1 ] -
           -agentOf->[person : super ],
           -objOf->[money]
-------------------------
An other definition (y/n) ?
```

aminePlatform.applications.ace.csAce2cg.aceRule2cg. ACERule2CG

A class that maps the rules formulated in ACE to CG:

```
INFO: Loading ontology ...
12 oct. 2013 19:55:32 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Ontology loaded in 28 seconds and 313 milliseconds.
NB : This program is case sensitive, please don't begin with an uppercase letter
except for Peron name or unknown word.


Antecedent :
a person has the money
Consequence :
the person is rich
12 oct. 2013 19:55:42 aminePlatform.interfaces.ace.util.ACELogger log
INFO: Analyzing ACE Text : If a person has the money then the person is rich. ...
Antecedent :
[have : *p1 ] -
           -agentOf->[person : *c1 ],
           -objOf->[money]
Consequence :
[person : ?c1 ]-propertyOf->[rich]
--------------------------
An other rule (y/n) ?
```

References

– Nasri M., KAbbaj A., Bouzoubaa K., Integration of a controlled natural language in an intelligent systems platform, Journal of Theoretical and Applied Information Technology October 2013, 252-262 [pdf].

– John F. Sowa, Conceptual Graphs, Handbook of Knowledge Representation, 2008, 213-237.