Integration of the controlled language ACE to the Amine Platform

Mohammed Nasri[1], Adil Kabbaj[2], Karim Bouzoubaa[3]

[1]EMI, Rabat, Morocco,
mohammed.nasri@gmail.com

[2]INSEA, Rabat, Morocco, B. P 6217
akabbaj@insea.ac.ma

[3]EMI, Rabat, Morocco
Karim.bouzoubaa@emi.ac.ma

Abstract. This paper presents the integration of the controlled language ACE (Attempto Controlled English) to Amine platform. Since the parser engine of ACE (ACE Parser Engine or APE) generates a DRS structure (Discourse Representation Structure), we have developed a mapping from DRS to CG (DRS2CG) which produces a CG equivalent to the DRS produced by APE. Through this mapping and this integration of ACE into Amine, Amine users can use controlled language to express their knowledge or specifications, instead of having to express them in CG directly.

Keywords: Natural language processing controlled language, ACE, DRS, CG, and DRS to CG mapping.

1 Introduction

This paper presents the integration of the controlled language ACE [9, 11] into the Amine platform [1, 2][1]. Amine is an open source software dedicated to the development of intelligent systems and agents. Amine provides Conceptual Graphs (CG) as its main knowledge representation formalism [7, 8].

Instead of formulating knowledge directly in CG, it would be easier and more convenient to formulate it in natural language. For this purpose, Amine should be extended by adding a Natural Language Processing component.

Given the numerous complexities and ambiguities of natural language, many researchers have chosen to focus on one subset of natural language[2], conventionally called "controlled language". Among these researchers is the Attempto group[3].

Controlled Natural Languages are subsets of natural languages whose grammars and dictionaries have been restricted in order to reduce or eliminate both ambiguity and complexity.

1 http://sourceforge.net/projects/amine-platform
2 http://sites.google.com/site/controllednaturallanguage/
3 http://attempto.ifi.uzh.ch

In our study, we opt for the controlled language ACE (Attempto Controlled English). ACE is a rich subset of Standard English designed to serve as knowledge representation language. ACE allows users to express texts precisely, and in the terms of their respective application domain [12, 13].

The Attempto group worked on various researchs and projects based on ACE such as the syntactic-semantic parser APE (ACE Parser Engine) [9] which extracts the ACE sentence semantics and represents it in the DRS formalism (Discourse Resource Structure) [10, 4].

In order to integrate ACE/APE with Amine, we developed a mapping between DRS and CG called DRS2CG. A sentence in ACE would be handled by APE generating a DRS structure from which DRS2CG generates a CG (Figure 1).

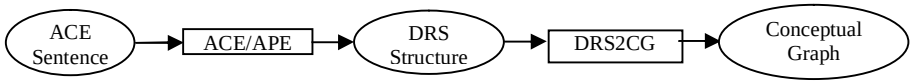


Figure .1 Natural Language Processing Component in Amine Platform.

This paper is organized as follows: Section 2 introduces briefly the controlled language ACE and DRS formalism, section 3 presents DRS2CG mapping for simple and composite sentences with some examples, the fourth one describes some applications of this mapping. The article ends with a conclusion and some future works

2 ACE and DRS

2.1. ACE

ACE (Attempto Controlled English) is a language specifically designed to write specifications. It is a controlled natural language, i.e. a subset of English with a domain specific vocabulary and a restricted grammar in the form of a small set of construction and interpretation principles. ACE allows representing specifications in simple or compound sentences. It also helps to formulate questions or queries.

Any ACE instruction consists of sentences, a sentence consists of words or other sentences.

Each sentence is characterized by a function that defines its role in a given context (like subject, complement, etc.) and a form defined as: negative sentence, positive sentence, verbal sentence, etc.

A set of optional elements called "modifiers" can be added to a sentence so as to give additional information; ACE accepts noun modifiers (adjectives, relative sentences, propositional phrases, possessive nouns and appositions) and verb modifiers (adverbs and prepositional phrases).

In ACE, users can build composite sentences from simple sentences, or composite phrases from simpler phrases, with the help of so-called constructors. ACE provides four different types of constructors: coordinators, subordinators, quantifiers and negators.

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

2.2. DRS

The ACE parser translates an ACE text unambiguously into a DRS representation [9]. It consists of a set of elements: "object", object "properties", possession "relations", "predicate", "modifier_adv" / "modifier_pp" to give more information to the predicate, "has_part" to specify that an object belongs to a group and "query".

Due to space limitation, we give briefly the structure of each element of DRS in the following parts, for more details, please refer to DRS technical report [10]:

- Object: Object (Ref, Noun, Quant, Unit, Op, Count)
- Property
  Property (Ref1, Adjective, Degree)
  Property (Ref1, Adjective, Degree, Ref2)
  Property (Ref1, Adjective, Ref2, Degree, CompTarget, Ref3)

- Relation: Relation (Ref1, of, Ref2)
- Predicate
  Predicate (Ref, Verb, SubjRef)
  Predicate (Ref, Verb, SubjRef, ObjRef) for a transitive verb
  Predicate (Ref, Verb, SubjRef, ObjRef, IndObjRef) for a ditransitive verb

- Modifier
  modifier_adv (VerbRef, Adverb, Degree) for verb modifier
  modifier_pp (VerbRef, preposition, ObjRef) for noun modifier

- Has_part: has_part (GroupRef, MemberRef)
- Request: Query (Ref, QuestionWord)

Here are some examples of structures with the DRS corresponding ACE sentences:

- *Example 1:* « a card is green. »:
  object (A, card, countable, na, eq, 1), property (B, green, pos), predicate (C, be, A, B).

- *Example 2:* « A customer has at least 2 cards that are valid. »:
  object (A, customer, countable, na, eq, 1), object (B, card, countable, na, geq, 2), property (C, valid, pos), predicate (D, be,B, C), predicate (E, have, A, B).

- *Example 3:* « A customer enters a card quickly and manually in a bank in the morning. »:
  object (A, customer, countable, na, eq, 1), object (B, bank, countable, na, eq, 1), object (C, card, countable, na, eq, 1), predicate (D, enter, A, C), modifier_pp (D, in, E), modifier_pp (D, in, B), modifier_adv (D, manually, pos), modifier_adv (D, quickly, pos), object (E, morning, countable, na, eq, 1).

3 DRS2CG mapping

3.1. Simple sentences

DRS2CG mapping uses the DRS structure so as to extract all the semantic elements and generate the corresponding conceptual graph. As mentioned by John Sowa, "Kamp's DRS notation is isomorphic to Peirce's existential graphs (EG), and conceptual graphs are a typed version of EGs" [6].

DRS mapping to CG is done as follows:

- Mapping of Object construct :

DRS: Object(Ref,Noun,Quant,Unit,Op,Count)
CG:
[Noun] -quant->[Quant]
-count->[number]<-Op-[integer : "Count"]

In a single object case (Quant=countable, op=eq et Count =1) we generate the concept [noun].

- Mapping of Property construct :

DRS: property(Ref,Adjective,Degree)
CG:
[Object(referenced by Ref)]-property->[Adjective]

DRS: property(Ref1,Adjective,Degree,Ref2)
CG:
[Adjective] -
<-property - [Object(referenced by Ref1)],
-target->[Object(referenced by Ref2)],
-opComp->[Degree],

DRS: property(Ref1,Adjective,Ref2,Degree,CompTarget,Ref3)
CG:
If CompTarget is subj, this corresponds to:

[Adjective] -
<-property - [Object(referenced by Ref1)],
-target->[Object(referenced by Ref2)],
-opComp->[Degree],
<- property-[Object(referenced by Ref3)]

Otherwise
[Adjective] -
<-property - [Object(referenced by Ref1)],
-target1->[Object(referenced by Ref2)],
-opComp->[Degree],
-target2->[Object(referenced by Ref3)]

- Mapping of Relation construct :

DRS: relation(Ref1,of,Ref2)
CG:

[Object(referenced by Ref1)] - poss ->[Object(referenced by Ref2)] if Res2 refers to an object, and
[Object(referenced by Ref1)] - attr ->[Object(referenced by Ref2)] if ref2 refers to an attribute.

- Mapping of Predicate construct :

DRS: predicate(Ref,Verb,SubjRef)
CG:
[Object(referenced by SubjRef)]<-Agent-[Verb]

DRS: predicate(Ref,Verb,SubjRef, ObjRef)
CG:
[Verb] –
-Obj->[Object(referenced by ObjRef)]
-Agent->[Object(referenced by SubjRef)]

DRS: predicate(Ref,Verb,SubjRef, ObjRef, IndObjRef)
CG:
[Verb]-
-Agent ->[Object(referenced by SubjRef)],
-Obj->[Object(referenced by ObjRef)],
-Dest ->[Object(referenced by IndObjRef)]

- Mapping of Modifier construct:

DRS: modifier_adv(VerbRef,Adverb,Degree)
CG:
[Verb(referenced by VerbRef)]->manner->[Adverb]

DRS: modifier_pp(VerbRef,preposition,ObjRef)
CG:
[Verb(referenced by Ref)]-loc/time ->[preposition]

- Mapping of Has_part construct:

DRS: has_part(GroupRef,MemberRef)
CG: [Group] <- partOf - [Member]

- Mapping of Query construct:

DRS: query(Ref,QuestionWord)
CG: [Location : x], [Time : x], [Manner : x] or [Object : x]

The mapping of DRS structure to CG is based on these correspondences. Here is an example of the mapping of a simple sentence.

ACE: John gives Mary at least 2 cards in the bank carefully.
CG:

```
[give : *p3 ] -
        -agent->[Person :John ],
        -obj->[card : *p1 ] -
                    -quant->[countable],
                    -count->[number : *p2 ]<-greaterThanORequalTo-[number : 2 ];
        -dest->[Person :Mary ],
        -loc->[bank],
```

-mann->[carefully]

The next section presents the DRS2CG mapping for the composite sentences.

3.2. Composites sentences

DRS2CG mapping also handles ACE composite sentences. Recall that ACE builds composite sentences using four constructors: coordination, subordination, quantifiers and negators.

In addition to simple and composite sentences, DRS2CG handles questions, commands, possibility, necessity, admissibility, recommendation and provability ACE sentences.

- Coordination: ACE distinguishes between two types of coordination: conjunction "AND" and disjunction "OR".
  In the conjunction case, DRS2CG generate the following CG:
  [A] – and ->[B]
  And in the disjunction case:
  [A] – or ->[B]
  "A" and "B" are the CGs corresponding to coordinated sentences ("A and B", "A or B").

- Subordination: Subordination means combining elements of unequal syntactic status. In APE, two types of subordination are supported: relative sentences and if-then sentences.
  The relative sentence is treated as a separate sentence conjuncted to the rest of the sentence. The ACE sentence "John who is a customer waits" is traited as: "John is a customer and John waits". In this case, DRS2CG interface is based on the conjunction principle so as to generate the CG.
  However, the if-then sentence is treated as an implication element that must contains two elements, the first one represents the antecedent and the second the consequence.
  The corresponding CG to an if-then statement generated by DRS2CG is:
  [Ant] – imply ->[Conseq]
  "Ant" and "Conseq" are the CGs corresponding to Antecedent and the Consequence sentences.

- Quantification: Quantified sentences are used to make assertions about the number of persons or objects that are involved in an event or state, the quantification is translated to an if-then sentence, for example the quantification "Every card is valid." is translated to "if there is a card then this card is valid.".
  In this case, DRS2CG interface is based on the if-then principle so as to generate the CG (previous part).

- Negation: The negators are used to express that something is not happening, not true or not the case. APE maps a negative sentence to a Negation element in which the sentence is mapped.

Our mapping DRS2CG generates the following CG for a negative sentence:

[Prop] – property ->[veracity: "false"]

"Prop" is the CG corresponding to the sentence on which we are applying the negation.

- Question: DRS2CG supports Yes/No, Where, When, Who, which and How questions. APE maps a question sentence to a Question element in which the sentence is mapped:

[Prop] – property ->[question: questionTypesSet]

questionTypesSet is a set of question types, example : {who, what}.

The same mapping is used in the next sentences types: Command, Possibility, Necessity, Recommendation, Admissibility and Provability.

- Command: The DRS2CG generated CG for a command sentence is:

[Prop] – property ->[Command]

- Possibility: The DRS2CG generated CG for a possibility sentence is:

[Prop] – property ->[Possibility]

- Necessity: The DRS2CG generated CG for a necessity sentence is:

[Prop] – property ->[Necessity]

- Recommendation: The DRS2CG generated CG for a recommendation sentence is:

[Prop] – property ->[Recommendation]

- Admissibility: The DRS2CG generated CG for a admissibility sentence is:

[Prop] – property ->[Admissibility]

- Provability: The DRS2CG generated CG for a provability sentence is:

[Prop] – property ->[Provability]

Here are some examples of the mapping of composite sentences.

– *Example 1 (Conjunction):* "John eats an apple and Mary waits."

CG:

```
[cg : [eat : *p1 ] -
        -agent->[Person :John ],
        -obj->[apple]
    ]-and->[cg : [Person :Mary ]<-agent-[wait]]
```

– *Example 2 (If-then):* "if the code is valid then the machine accepts the card."

CG:

```
[ant : [code]-property->[valid]
    ]-imply->[conseq : [accept : *p1 ] -
                    -agent->[machine],
                    -obj->[card]
            ]
```

– *Example 3 (Quantification):* "Every card is valid."

CG:

```
[ant : [card : *c1 ]
    ]-imply->[conseq : [card : ?c1 ]-property->[valid]   ]
```

– *Example 4 (Question):* "Is the card valid?"

CG:

```
[prop : [card]-property->[valid]
    ]-property->[question: yes_no ]
```

– *Example 5 (Necessity):* "A customer must enter a card."

CG:

```
[prop: [enter : *p1 ] -
            -agent->[customer],
            -obj->[card]
      ] -property->[necessity]
```

– *Example 6 (Provability):* "A card is not provably valid."

CG:

```
[prop : [prop : [card]-property->[valid]
        ]-property->[provability]
    ]-property->[vericity : "false" ]
```

4 ACE2CG integration into the Amine Platform

Currently, four actions have been realised toward a full use of ACE2CG and its integration with Amine Platform: a) the development of an ACE2CG Graphical user interface, b) the formulation of Conceptual Structures in ACE, c) the development of a simple Text Analysis and Question/Response System, d) Prolog+CG rules generator from specifications written in ACE. These actions are described in the next sections.

4.1. ACE2CG graphical user interface

We have developed a graphical interface which illustrates ACE/APE and ACE2CG mapping (figure 2). This GUI allows user to load his ontology, then edit his ACE text. The activation of the button "Do Syntactico-Semantic Analysis" activates the ACE/APE which produces a DRS structure that is mapped to an equivalent CG by ACE2CG. The result (a CG) is shown in the bottom panel (Figure 2).

4.2. Conceptual Structures in ACE

Amine uses CG to represent knowledge. Also, Amine enables the construction and edition of ontologies composed of conceptual structures (concept type definition, relation type definition, canon, individual, schema/situation, and rules). With ACE/APE and DRS2CG, Amine offers now the possibility to express knowledge and conceptual structures directly in controlled natural language (ACE), instead of forcing user to express his knowledge in CG according to a specific notation.

The first application of ACE2CG (ACE-DRS2CG) interface was its integration into Amine Platform and the formulation of conceptual structures in ACE instead of CG.

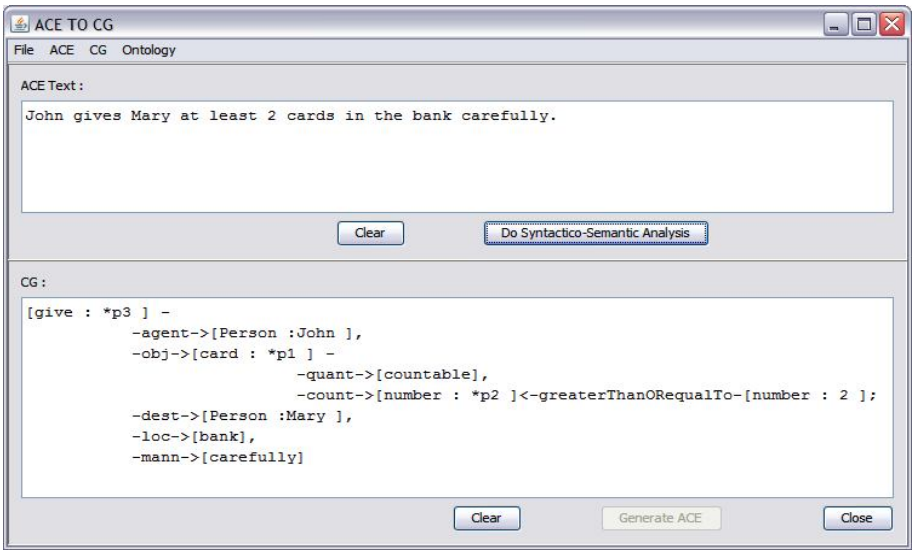


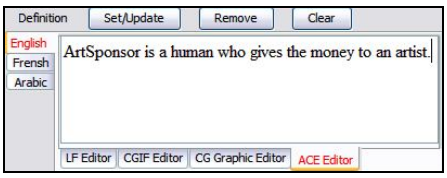
Figure .2 ACE2CG GUI using sample

A new tab (ACE Editor) has been added to allow user editing text in ACE language instead of CG formalism.

Amine users are therefore able to formulate conceptual structures using Linear Form, CGIF Form [5], Graphical representation or ACE controlled language. Here are some examples that illustrate this new possibility:

- Type Definition and Canon:

We can edit the canon or definition of the type in ACE, on switching the tab, the CG corresponding to ACE definition is generated in different representations (figures 3.a, 3.b, 4.a and 4.b).



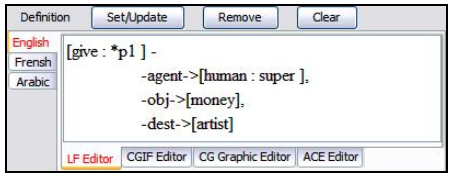
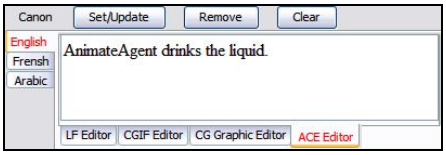


Figure .3.a: Definition of ArtSponsor in ACE **Figure .3.b:** Definition of ArtSponsor in CG LF



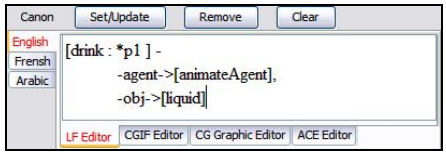


Figure .4.a: Canon of AnimateAgent in ACE **Figure .4.b:** Canon of AnimateAgent in CG LF

- Rules:

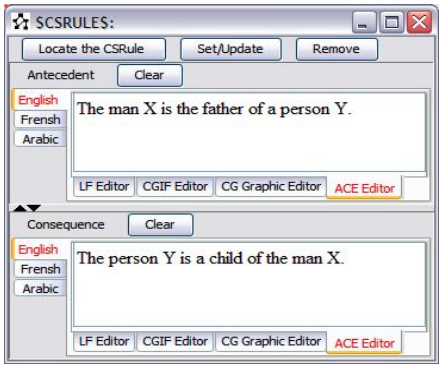


Figure .5.a: Rule in ACE

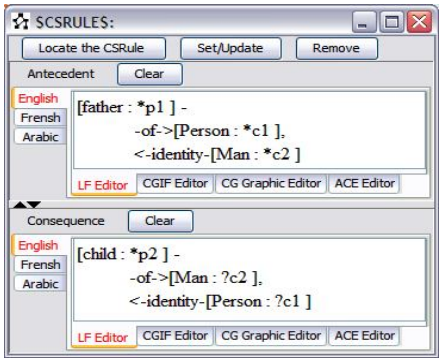


Figure .5.b: Rule in CG LF

- Situations:

Figure .6.a: Situation in ACE

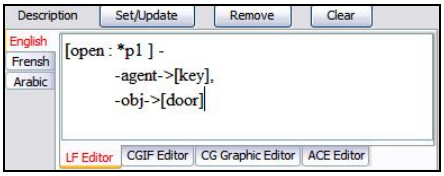


Figure .6.a: Situation in CG LF

4.3. Text Analysis and Questions Answering System (TAQAS)

The third application of ACE2CG mapping is a simple Text Analysis and Question Answering System based on ACE.

TAQAS is a system that analyses a text given by users and generates the CG corresponding to its semantic.

The user can then ask TAQAS questions about the meaning conveyed by the text. TAQAS analyses the question and generates its CG and through cg operations (like subsume) TAQAS extracts the response CG.

TAQAS can actually trait all allowed question types in APE (Which, What, Who, How, Where and When).

TAQAS was integrated into Amine too with a user friendly interface (figure 7).

Here is an example of question answering for the same ACE text: “John is in the bank in the morning. John gives Mary a card carefully. The code is correct and the bank accepts the card and Mary is happy.”

Please note that TAQAS is just a simple prototype that illustrates what can be done using the mapping of ACE to CG; the negation, recommendation, necessity and the others modality sentences are not supported. It is also noteworthy that the system is only looking for information in the base and does not seek its truth.

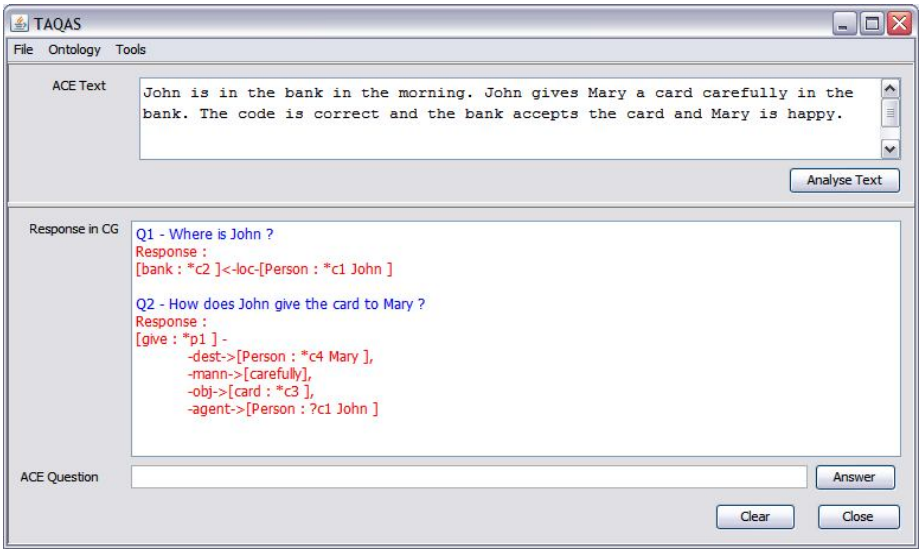


Figure .7 TAQAS questions analysis example

4.4. ACE as an executable specification language, with ACE2CG and Prolog+CG

Schwitter proposes to use ACE as an executable specification language [12, 14, 15]. As an example, he specifies a library database application as a set of rules expressed in ACE [15]. The result of the syntactico-semantic analysis of the rules correspond to DRS structures that are reformulated in Prolog.

With ACE2CG and Prolog+CG [3], we obtain a more simple and direct analysis of executable specification using ACE; the analysis of the rules expressed in ACE produces Prolog+CG rules, instead of Prolog rules. Below are the two Prolog+CG rules that result from the mapping of two rules from a specification in ACE:

- "If a member adds a copy of a book to the library and no entry of the book exists in the catalog then the member creates an entry and the member enters the id of the copy and the copy is available." (Figure 8).

- "If a copy of a book is checked-out to a borrower and a member returns the copy then the copy of the book is available." (Figure 9).

It should be noted that if the same object has been introduced and is referred in the same rule, ACE2Prolog+CG generates a variable for it, otherwise if it is no referred.

```
 ACE Text :
  If a member adds a copy of a book to the library and no entry of the book exists in the catalog
        then the member creates an entry
          and the member enters the id of the copy
          and the copy is available.


 Prolog+CG rule :
 [cg : *p1  [id : *p3 ] -
                   <-attr-[copy :  c4 ],
                   <-obj-[enter : *p4 ]-agent->[member :  c3 ]
            ] -
-and->[cg : [copy :  c4 ]-property->[available]
            ],
<-and-[cg : [create : *p2 ] -
                         -agent->[member :  c3 ],
                         -obj->[entry]
            ] :-
       [plus : *p1 ] -
          -agent->[member :  c3 ],
          -obj->[copy :  c4 ]<-poss-[book :  c2 ],
          -to->[library],
       [ant : [entry :  c1 ]<-poss-[book :  c2 ]
       ]-imply->[conseq : [prop : [exist : *p1 ] -
                                      -agent->[entry :  c1 ],
                                      -in->[catalog]
                           ]-property->[vericity : false ]
                  ].
```

Figure 8 ACE2Prolog+CGRule example 1

```
ACE Text :
If a copy of a book is checked-out to a borrower and a member returns the copy
       then the copy of the book is available.


Prolog+CG rule :
[copy :  c1 ]-property->[available] :-
      [copy :  c1 ] -
         -property->[checked-out],
         -to->[borrower],
         <-poss-[book],
      [return : *p1 ] -
           -agent->[member],
           -obj->[copy :  c1 ].
```

Figure 9 ACE2Prolog+CGRule example 2

Conclusion

In this paper, we presented our work about natural language processing into Amine, especially the mapping from ACE to CG, which allows for processing language specifications written in controlled language ACE to generate conceptual graphs corresponding to their semantics.

This work has been integrated into Amine platform in order to allow users use ACE (which is more convenient) instead of the CG formalism. Thus, as first concrete implementation, the user can now formulate conceptual structures in ACE.

ACE2CG can be the basis of several works, particularly those based upon CGs, some of them have been outlined above as an illustration (syntactico-semantic analysis of Text, Q/A and information retrieval, executable specification), but this list is far from being complete. For instance, this work can be used also in many semantic web applications.

Future Works

We are concerned by at least four axes of research:

- Generation of ACE from CG. This can be used in Amine CG editor (generates an ACE formulation for a given CG), in Question/Answering, in translation systems and in many other applications.
- The development of a "controlled Arabic language", we could then develop an Arabic-English translation system based on semantic.
- The development of the use of ACE/CG as an executable specification.
- The extension of ACE in order to support more sentence forms.

References

1. A. Kabbaj, An Overview of Amine, in P. Hitzler and H. Schärfe (Eds.), Conceptual Structures in Practice, pp. 321-348, Taylor and Francis Group, Chapman and Hall/CRC, 2009.
2. A. Kabbaj, Development of Intelligent Systems and Multi-Agents Systems with Amine Platform, in H. Scärfe, P. Hitzler, P. Ohrstrom (Eds.), Conceptual Structures: Inspiration and Application, LNAI 4068, pp. 286-299, Springer 2006.
3. A.Kabbaj, PROLOG+CG version 2.0, user's manuel.
4. H. Kamp and U. Reyle, From Discourse to logic, Kluwer Academic Publishers, 1993.
5. J. F. Sowa, Conceptual Graphs for Representing Conceptual Structures, in P. Hitzler and H. Schärfe (Eds.), Conceptual Structures in Practice, pp. 101-136, Taylor and Francis Group, Chapman and Hall/CRC, 2009.
6. J. F. Sowa, Conceptual Graphs, in F. van Harmelen, V. Lifschitz, and B. Porter, eds., Handbook of Knowledge Representation, Elsevier, 2008, pp. 213-237.
7. J. F. Sowa, Conceptual Structures: Information Processing in Mind and Machines, Addison-Wesley, Reading, MA, 1984.
8. J. F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks/Cole Publishing Co., CA, 2000.
9. Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In Cristina Baroglio, Piero A. Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, Reasoning Web, Fourth International Summer School 2008, Lecture Notes in Computer Science 5224, pages 104–124. Springer, 2008.

10. Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn. Discourse Representation Structures for ACE 6.6, Technical Report ifi-2010.0010, Department of Informatics, University of Zurich, 2010.
11. Norbert E. Fuchs, Stefan Höfler, Kaarel Kaljurand, Fabio Rinaldi, and Gerold Schneider. Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines. In Norbert Eisinger and Jan Maluszynski, editors, *Reasoning Web, First International Summer School 2005*, Msida, Malta, July 25–29, 2005, Revised Lectures, Lecture Notes in Computer Science 3564. Springer, 2005.
12. N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto Controlled English — Not Just Another Logic Specification Language, In P. Flener, editor, Logic-Based Program Synthesis and Transformation, 8th International Workshop LOPSTR'98, LNCS 1559. Springer-Verlag, 1999.
13. N. E. Fuchs, U. Schwertel, and S. Torge, Controlled Natural Language Can Replace First-Order Logic, 14th IEEE International Conference on Automated Software Engineering, pp. 295-298, Society Press, 1999.
14. R. Schwitter, Controlled English for Requirements Specifications, PhD thesis, Department of computer science, University of Zurich, 1998.
15. R. Schwitter, N. E. Fuchs. Attempto - From Specifications in Controlled Natural Language towards Executable Specifications, GI EMISA Workshop, Natürlichsprachlicher Entwurf von Informationssystemen, Tutzing, Germany, May 1996.