Standard CGIF interoperability in Amine

KABBAJ, A., LAUNDERS, I. and POLOVINA, S. <http://orcid.org/0000-0003-2961-6207>

Published version

KABBAJ, A., LAUNDERS, I. and POLOVINA, S. (2009). Standard CGIF interoperability in Amine. In: Fourth Conceptual Structures Tool Interoperability Workshop (CS-TIW 2009) at the 17th International Conference on Conceptual Structures, Moscow, Russia, July 25 2009.

Standard CGIF Interoperability in Amine

Adil Kabbaj[1], Ivan Launders[2] and Simon Polovina[3]

[1]INSEA, Rabat, Morocco
akabbaj@insea.ac.ma
[2]BT Global Services, PO Box 200, London, United Kingdom
ivan.launders@bt.com
[3]Cultural, Communication & Computing Research Centre (CCRC)
Sheffield Hallam University, Sheffield, United Kingdom
s.polovina@shu.ac.uk

Abstract. The adoption of standard CGIF by CG tools will enable interoperability between them to be achieved, and in turn lead to the interoperability between CG tools and other tools. The integration of ISO Common Logic's standard CGIF notation in the Amine platform is presented. It also describes the first steps towards full interoperability between the Amine CG tool (through its Synergy component) and CharGer, a representative CG tool that supports similar interoperability and for process (or 'active') knowledge as well as declarative knowledge. N-adic relations are addressed as well as semantic compatibility across the two tools. The work remarks on the successes achieved, highlighting certain issues along the way, and offering a real impetus to achieve interoperability.

1 Introduction

The adoption of a standard that will enable the actual interoperability of Conceptual Graphs (CG) tools both between them and other tools has been a long-standing interest of this community. Through such means the CG tools themselves will become more widely adopted as there is less 'lock in' to the limitations of a particular tool, and in turn collectively bring the power of CG into wider arenas such as the Semantic Web. It is in this context that the standard for the Conceptual Graph Interchange Format (CGIF), a fully conformant dialect of ISO Common Logic (CL) [1], describes a set of transformation rules. CGIF is a representation for conceptual graphs intended for transmitting CG across networks and between IT systems that use different internal representations. The design goal for CGIF was to define a minimal abstract syntax and notation to achieve:

- a proper foundation with translations to Knowledge Interchange Format (KIF) and permits extensions to be built upon;
- a rapid interchange of CG between CG tools.

Annex B [1] provides the CGIF syntax[1] and its mapping to CL. The Amine software platform [3,4] has accordingly been developed towards interoperability both in terms of syntax and semantics.

2 Integrating Standard CGIF in Amine for Interoperability

The integration of the Standard CGIF notation in Amine [3,4] (since Amine 6.0) involves many other integrations/additions (like a mapping for actors and a mapping for n-adic relations with n > 2), as follows.

2.1 From "Amine CGIF" to Standard CGIF notation

In previous versions, Amine adopted a proper subset of CGIF that we refer to "Amine CGIF". There are certain differences between the Amine CGIF and Standard CGIF notation. Amongst them there is the use of the "pseudo-designator" in "Amine CGIF". To explain, in Standard CGIF the coreferent has two functions:

- It is used in the CGIF notation (and CG's Linear Form (LF) notation) to specify occurrences of the same concept in the same CG. Such a coreferent exists neither in the Display form, nor in the internal representation of the CG.
- It is used to specify coreferent links between concepts that belong to embedded CG.

The Pseudo-designator is used in Amine CGIF to play the first function of a coreferent. Instead of pseudo-designator, standard CGIF notation makes a direct use of a coreferent. For instance, instead of the pseudo-designator #1 in this example:

```
[Work #1] [Person: Jane] [House : myHouse]
(agnt #1 Jane)  (near  #1  myHouse)  (ageOf  Jane  [Age = 30])
(poss  myHouse  Jane)
```

Standard CGIF notation [1] will use a coreferent:

```
[Work *p] [Person: Jane] [House : myHouse]
(agnt ?p Jane)  (near  ?p  myHouse)  (ageOf  Jane  [Age:30])
(poss  myHouse  Jane)
```

In Amine CGIF the designator and descriptor represents two different fields of a concept. In standard CGIF notation, coreferent, designator and descriptor share the same and one field; namely the field after the concept type. In Amine CGIF, a separator is used to differentiate between the designator and the descriptor. In standard CGIF notation, no separator is used to introduce and differentiate between the designator and the descriptor. The separator ":" is optional in Standard CGIF and there is great flexibility in ordering the coreferent, designator and descriptor; a coreferent can be placed before or after the designator and/or the descriptor.

[1] In describing the syntax of CGIF, B.1.2 [1] uses the Extended Backus-Naur Form (EBNF) notation as referenced in ISO/IEC 14977:1996. However the classic Sowa [5] examples are used in the paper as referenced in the standard for CGIF [1].

To assure backward compatibility with earlier versions of Amine 6.0 (i.e. that use just the Amine CGIF notation), Amine 6.0 adopts a CGIF syntax that integrates standard CGIF notation and Amine CGIF notation. A CG that was stored in Amine CGIF notation is thus still readable by Amine, but it will be reformatted into standard CGIF notation (i.e. a new save of the CG, in CGIF, will use standard CGIF notation).

Here is the syntaxic grammar of a concept in Standard CGIF notation in Amine (a complete description of the grammar of Standard CGIF notation in Amine is provided in the Amine Web Site[2]):

```
Concept ::= "[" [Comment] Conc_Type
[ParameterModes (generate in EndComment)] [CoReferent]
     [":"] [CoReferent] {Designator} [CoReferent] ["="] [Descriptor]
     [CoReferent] ["$" State (generate in EndComment)]
["&" Point (generate in EndComment)] [EndComment]     "]"
Comment ::= "/*", {(character-"*") | ["*", (character-"/")]}, ["*"],
"*/"
EndComment ::= ";", {character - ("]" | ")")}
```

Note that a concept can start with a comment and terminate with an end comment. A coreferent can be stated before or after the separator ":", before or after a designator, and before or after a descriptor. Separators ":" and "=" are optional. Parameter modes (appropriate to the Synergy component in Amine), States (appropriate to Synergy), and Point (appropriate to the CG display form) are also optional, and they can be specified inside the concept or in the end comment. Note also that the above syntaxic rule is used to define the syntax of concepts in both CGIF and LF notations. Thus, with Amine 6.0 we adopt the above syntaxic rule for the representation of concept in both LF and CGIF forms.

Example #1 To illustrate, Sowa's classical example ("John Goes to Boston by Bus") is shown in the LF notation, Standard CGIF, and Display Form (Figure 1):

LF notation:

```
[Go]-
   -instr->[Bus],
   -dest->[City :Boston ],
   -agnt->[Human :John ]
```

Standard CGIF notation (without graphical information):

```
[Go : *p1 ]
(instr ?p1 [Bus]) (agnt ?p1 [Human :John]) (dest ?p1 [City :Boston])
```

Standard CGIF notation (with graphical information):

```
[Go : *p1 ; & 235 89]
(instr ?p1 [Bus ; & 65 72]; (154 87)(235 97)(97 78)) (dest ?p1 [City
:Boston ; & 299 164]; (283 135)(253 106)(334 164)) (agnt ?p1 [Human
:John ; & 346 18]; (307 62)(255 89)(380 35))
```

(The graphical information, for both concepts and relations, are specified in the EndComment.)

[2] sourceforge.net/projects/amine-platform

Example #2 An example proposed by Sowa to illustrate nested CG:

LF notation:

```
[Believe] -
    -expr->[Human : "Tom" ],
    -thme->[Proposition : [Want] -
                    -expr->[Human : *x3 "Mary" ],
                    -thme->[Situation : [Marry] -
                                        -agnt->[Woman : ?x3 ],
                                         -thme->[Sailor]
                                ]
            ]
```

Standard CGIF notation:

```
[Believe : *p1 ]
(expr ?p1 [Human : "Tom" ]) (thme ?p1 [Proposition : [Want : *p2 ]
         (expr ?p2 [Human : *x3 "Mary" ])
         (thme ?p2 [Situation : [Marry : *p3 ]
                        (agnt ?p3 [Woman : ?x3 ]) (thme ?p3
[Sailor])
                                                    ])
                                        ])
```

2.2 Standard CGIF notation, Actors and "active concepts"

Concepts and relations capture declarative knowledge, but lack the capability for active or process knowledge. Actors[3] dynamically modify CG allowing active or process knowledge. Actors also provide the means to access external systems [2]. Actors constitute a standard element in CG theory, like concepts and relations. Standard CGIF [1] provides a precise syntax for their representation. Amine adopts another perspective: a concept can represent a static or a dynamic, executable entity (operation, function, process, etc.). Thus, concepts represent both “static concepts” and “dynamic concepts” (i.e. Actors). Amine aims to allow the integration of its Synergy engine with an ontology in order to perform actor-like processing[4].

As Amine integrates Standard CGIF notation (as of Amine 6.0), we can provide a mapping between “CG with Actors” and “Amine CG (with dynamic concepts)”. Thus a CG with Actors that is formulated in standard CGIF notation can be translated to an Amine CG. Hence, interoperability between Amine and other tools that use Actors is now possible, such as between Amine and the CharGer[5] CG software that has extensive support for Actors. Figure 2 is an example of this mapping: Figure 2.a shows CharGer visual display of a CG with Actors. CharGer provides the possibility to save a CG (with Actors) in standard CGIF (Figure 2.b). The file produced by CharGer can then be read and reformulated by Amine (Figure 2.c). However, to activate/execute the CG by Amine, semantic compatibility is required, in addition to the common use of standard CGIF notation. This point is discussed next.

---

[3] Not to be confused with Actors in the UML (www.uml.org)

[4] More information on the Synergy language can be found via the Amine website, amine-platform.sourceforge.net

[5] sourceforge.net/projects/charger

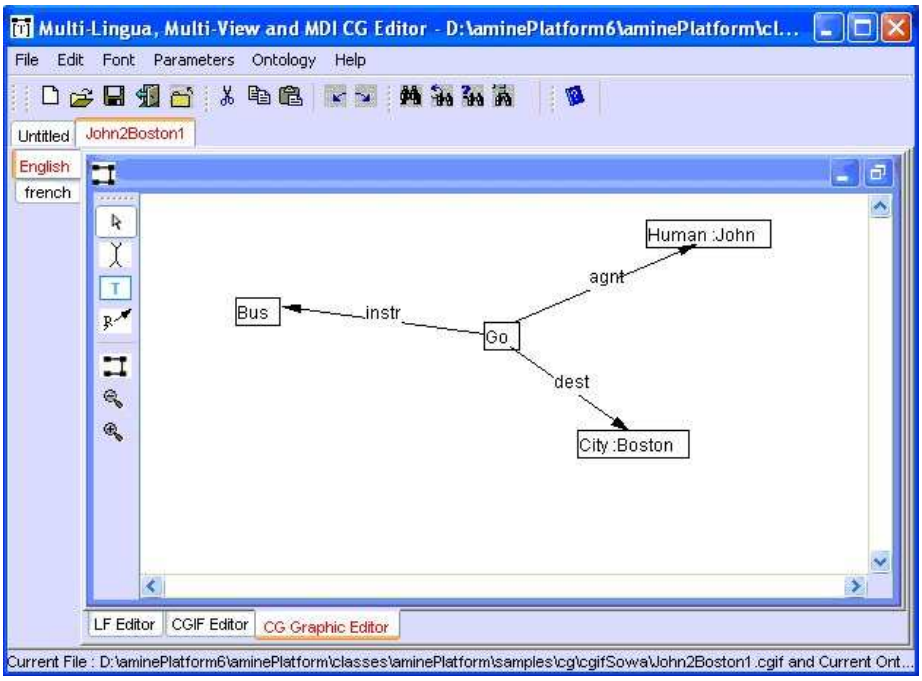

Figure 1: Diagram Form of a CG (in Amine)

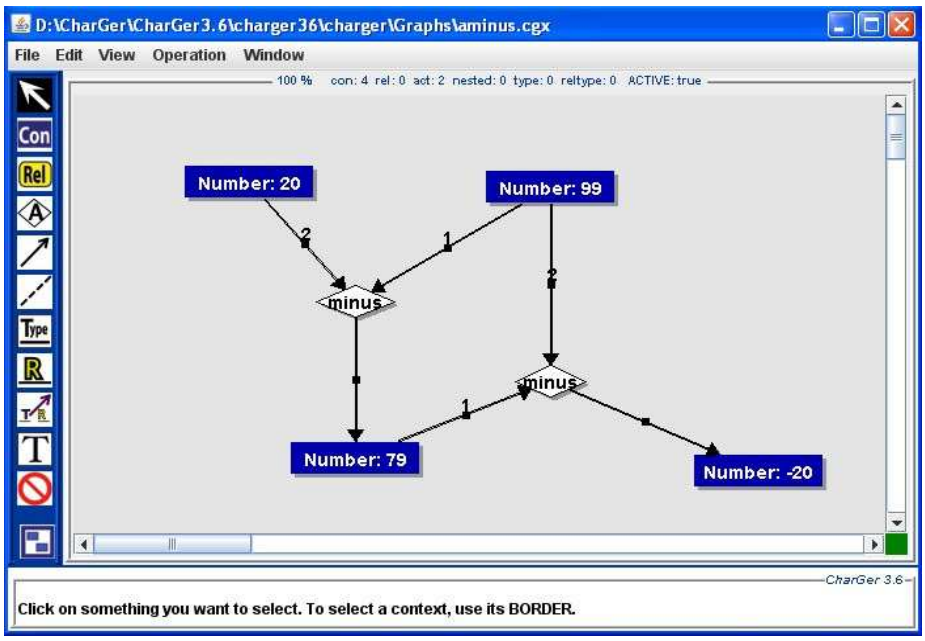

Figure 2.a: CharGer diagram notation of a CG with Actors

```
 [ /* <concept id="35979485995" owner="35979485989"> <type>
<label>Number</label> </type> <referent> <label>99</label>
</referent> <layout> <rectangle x="315.0" y="60.0" width="98.0"
height="25.0"/> <color foreground="255,255,255"
background="0,0,175"/> </layout> </concept> */ Number: 99]
...
 (/* <actor id="35979485996" owner="35979485989"> <type>
<label>minus</label> </type> <layout> <rectangle x="185.0" y="150.0"
width="60.0" height="25.0"/> <color foreground="0,0,0"
background="255,255,255"/> </layout> </actor> */ minus 99 20 | 79)
```

Figure 2.b: CharGer formulation of the above CG in Standard CGIF notation

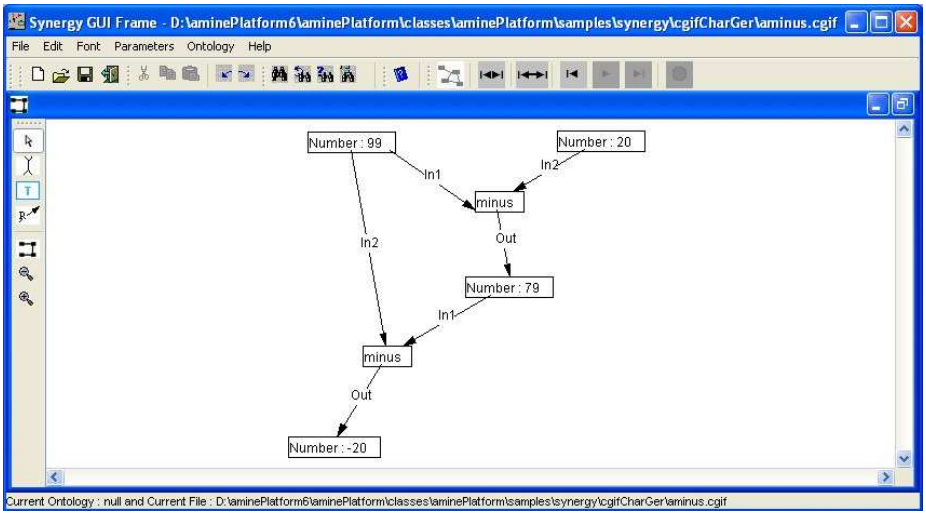

Figure 2.c: Amine's reformulation of a CG with Actors through standard CGIF translation

2.3 Standard CGIF, more than dyadic relations' reformulation in Amine

CG theory allows for conceptual relations that are more than dyadic, like the classical example of "Between" relation (Figure 3.a). Standard CGIF notation thus allows for the representation of more than dyadic relations. In Amine we consider dyadic relations only. However as Amine 6.0 integrates standard CGIF notation, we provide a mapping between "CG with more than dyadic relations" and "CG with dyadic relations only": CG with more than dyadic relations formulated in standard CGIF notation is thus translated to CG with dyadic relations only. This mapping is based on the representation of the more than dyadic relation by a concept and dyadic relations (Figure 3). With this mapping, we can now affirm that Amine handles n-adic relations (where n > 2) thereby providing interoperability with tools that allow the use of n-adic relations. Figure 3 provides the example of the "Between" relation (with "Betw" as a synonym). Figure 3.b shows how Amine 6.0 provides the possibility to use more than dyadic relation, through standard CGIF notation. But as illustrated by the diagram (Figure 3.c), the relation is reformulated in terms of dyadic relations.

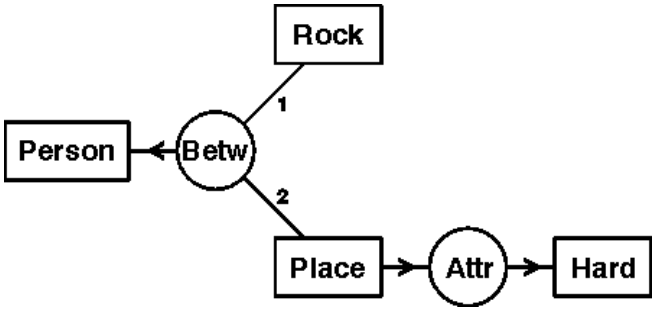

Figure 3.a: Triadic relation

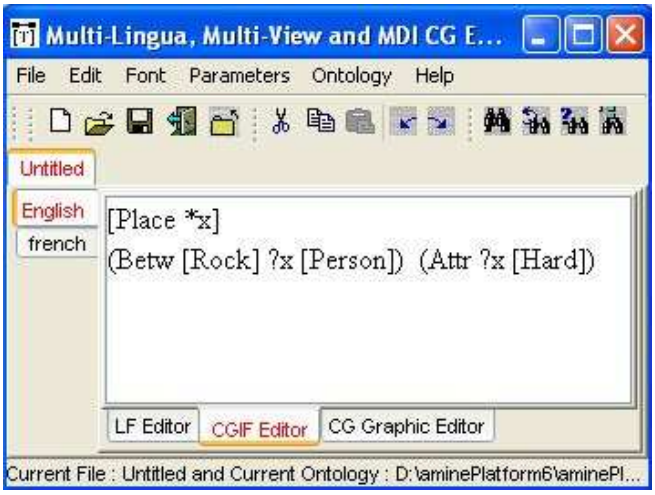

Figure 3.b: Triadic relation in standard CGIF

2.4 Semantic Compatibility

Sometimes, interoperability requires semantic as well as syntaxic compatibility. With the integration of standard CGIF in Amine 6.0, it is now possible for Amine to open and read CG that are created by CharGer. But this may not be enough; to enable a CG in CharGer to be executable by Synergy (and vice-versa), semantic compatibility may also be required. Synergy therefore provides semantic compatibility by:

- A mapping between CG with Actors (adopted by CharGer) and Amine CG with dynamic concepts (adopted by Amine);
- A mapping between CharGer's Actor identifiers with equivalent Synergy primitive operations. CharGer's Actor identifiers are added in Amine as synonyms for identifiers of Synergy primitive operations. For instance "plus" is added as a synonym of "Add", "minus as synonym of Sub", "multiply as synonym of Mul", etc.

- To guarantee CG that represent arithmetic expressions created by Synergy can be opened and activated by CharGer, CharGer's identifiers for arithmetic operations are preserved as the principal identifiers. Thus for instance, the Actor for addition will be expressed as [plus] instead of [Add].

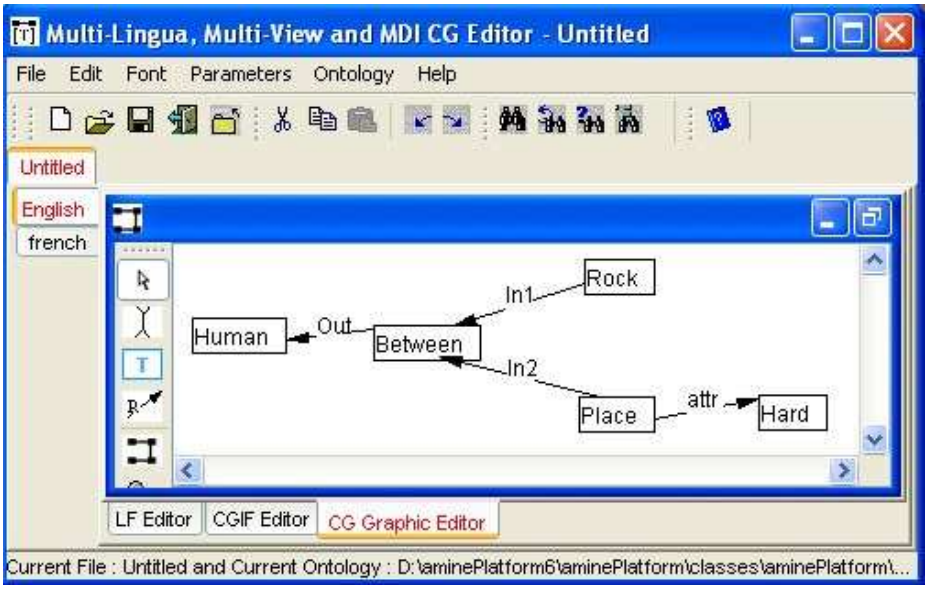

Figure 3.c: Reformulation of Triadic relation in Amine, through the CGIF translation

With the implementation of semantic compatibility between CharGer and Synergy, most of CharGer's CG can be activated and executed by Synergy. However since CharGer is equivalent to a subset of Synergy, certain CG with Amine/Synergy features cannot be activated by CharGer (such as Amine's CG that contain control relations, or CGs with defined concept types).

Thus, semantic compatibility remains partial because CharGer and Synergy consider a different semantic. Compatibility of the ontological level of the two tools is also partial: the ontology in CharGer corresponds to a type hierarchy, whilst ontology in Amine corresponds to a hierarchy of Conceptual Structures (Definition, Individual, Canon, Schema/Situation, Rule). Of course, a type can have no definition and no canon, and an individual can have no description. As the type hierarchy is a special case of Conceptual Structures hierarchy, any ontology used by CharGer can be automatically reformulated in Amine but not the inverse.

Semantic interoperability is also affected due to certain forms of CG that are presently unsupported in Amine; for instance "null" is not allowed as a concept type, a monadic expression is not allowed as a concept type, a monadic relation is not allowed (unless it can be re-expressed as a dyadic relation), and cardinality is not considered. Some of these features may be integrated in future versions of Amine, depending on how important it is considered by the CG and wider community that such functionality should be supported.

3 Concluding remarks

We have described the integration of Standard CGIF notation in Amine, and the impact of this integration on the interoperability between Amine and, by using CharGer as one exemplar, other (CG) tools. It was most heartening to experience interoperability actually being accomplished through the adoption of standard CGIF. This actual experience was not discouraged by CharGer version 3.6, being the version we worked with in this exercise, being able to store a CG in standard CGIF but had no support for reading a CG written in standard CGIF. Rather, such issues can be overcome (for example in this case in the next version of CharGer), and that our experience will act as a key encouragement.

We have also found ways to support semantic compatibility through our experiences with Amine and CharGer as we described, using Amine to recognise the differing semantics in another tool (CharGer) and adapt to it in a principled way.

Of course we need continued effort from all sides to achieve heightened interoperability between Amine, CharGer and across all CG tools. By raising some of the issues involved in doing so, but at the same time demonstrating the success we have had, we have provided a real impetus to achieve the interoperability that the Conceptual Structures community ultimately desires.

References

1. International Standards Organisation (2007) International Standards for Business, Government and Society ISO/IEC 24707:2007 Information technology -- Common Logic (CL): a framework for a family of logic-based languages. Last accessed on 17 June 2009 at http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39175
2. Delugach, Harry (2009) "A Practitioner's Guide to Conceptual Graphs", personal communication (available from the author)
3. Kabbaj A., *Development of Intelligent Systems and Multi-Agents Systems with Amine Platform,* in 15th Int. Conf. on Conceptual Structures, ICCS'2006, Springer-Verlag, 2006.
4. Kabbaj A., *An overview of Amine*, to appear in « Conceptual Structures in Practice: Inspiration and Applications », H. Schärfe, P. Hitzler (eds), Taylor and Francis Group, 2009.
5. Sowa, J., (1984). 'Conceptual structures: information processing in mind and machine', Addison-Wesley.