A First Step toward the *Knowledge Web*: Interoperability Issues among Conceptual Graph Based Software Agents Part I

Guy W. Mineau

Dept. of Computer Science, University Laval
Quebec City, Quebec, Canada, G1K 7P4
Tel: (418) 656-5189, Fax: (418) 656-2324
Mineau@ift.ulaval.ca

Abstract. As soon as Web documents embed knowledge in a format processable by computers, it is expected that knowledge-based services will be offered on-line, through the Web. These applications will query the Web to seek the information relevant to their task. Knowledge providers will host that knowledge and will make it available to these various applications. Agent technology is probably best suited to implement knowledge servers. This paper sketches how conceptual graphs (CG) based software agents could play the role of knowledge providers; as an example, it uses a situation where some agent must answer a query sent by some other agent. In doing so, this paper shows how interoperability problems between communicating conceptual graph based systems can be detected automatically. It also shows how semantic constraints can be used to implement *semantic filters*, filters required to control, on a semantic level, the information that is exchanged between communicating systems.

1 Introduction

Though the semantic Web seems to be quite an endeavor at the moment, one can already glance at the future beyond the semantic Web, when users and their applications will interconnect to a web of information-based *service* providers rather than to a web of *document* providers. Nodes in this Web will respond to queries on an information need basis rather than on a fetch (or ftp) command-like interaction as is done today through the identification of potentially interesting documents (Web pages), either directly through a Web browser or indirectly through a Web search engine. These nodes will act as *knowledge providers* rather than document providers, and will allow the first generation of the *knowledge Web* to be born. The knowledge Web will provide remote applications with the knowledge they require in order to carry out their tasks; it will offer real information-based *services* to the various software applications that will query it. Applications will rely on this library of accessible information-based services; they will be designed as collaborating software agents. This software reuse based design will surely decrease software development costs but will: a) increase testing costs, b) turn distributed applications into probabilistic applications where the probability of failure due to a non collaborating agent will most certainly be non negligible, and c) will require that time-sensitive applications be reengineered so that communication and collaboration time do not prevent them from providing timely services. Nevertheless, as telecommunication and computer hardware never stops to provide additional speed to software applications, and as on-line software brokerage repositories are part of a major effort of the industry to provide distributed software services over the Web [1], it is foreseeable that there will be both a need and an opportunity to design knowledge servers in a near future, as we already strive for the *knowledge Web*.

Toward that goal, Section 2 of this paper proposes conceptual graph-based software agents with regard to the task of query answering in such a setting. Then in Section 3 it presents the fundamentals of interoperability issues between CG-based systems that aim at communicating. In Section 4 it shows how particular interoperability conditions pertaining to the filtering of data (as identified in Section 3) can be fulfilled. Section 5 concludes by outlining the future directions of research that we intend to pursue.

2 CG-Based Software Agents as Knowledge Servers

In this paper we define a software agent as: *an automated task-oriented piece of software that has both reactive and proactive capabilities*. This entails that it is somewhat autonomous, that it can perceive a reality and decide to act upon it, that it can plan in order to get closer to achieving its goal, that it will seek to collaborate with other agents if it can not achieve its goal on its own, and therefore, that it can communicate with others.

For the sake of simplicity and to remain focussed on the topic of this paper, let us define a CG-based agent as being solely a knowledge server, that is, an agent whose main (and only) task is to provide answers to queries that it receives from other agents. Of course the spectrum of actions (of services) that an agent could render could be more elaborate. Such a simple agent could be seen as a CG system reacting to a query that it receives, deciding whether to answer it or not, and if so, in compliance with the various interoperability issues that condition the communication between itself and the agent where the query originated. Of course, its knowledge is contained in a CG knowledge base described, as usual, by a canon, which provides the fundamental elements: a set of partially ordered types T, a set of object representatives I (which could be either constants or variables), a conformance relation between types and object representatives $C : T \times I \rightarrow \{\text{false}, \text{true}\}$, and a set of semantic constraints H[1], all needed to restrict the universe of discourse to valid

[1] The reader should notice that H is based on the work of [2] and includes the canonical basis B normally defined as part of the canon of a CG system. Therefore, the canon that we present here is some extension of that of Sowa [3].

formulae, in order to avoid acquiring knowledge that could not be true of any model since it would violate the semantics of the domain. This tuple : <T,I,C,H> is called the canon of the system[2]. Based on the canon, knowledge describing the application domain can be asserted. We call the set of assertions A. Since A is the set of all asserted conceptual graphs, it forms the knowledge base accessible to the agent (to the system).

So each CG-based agent has a knowledge base into which all of its knowledge is stored. Of course, especially when modeling modalities or hypothetical worlds (as done when an agent builds a model of each agent with which it interacts), some form of memory segmentation/structuring may be required. We proposed such a structuring in [5] based on our previous work on contexts [6], but this clearly falls outside the scope of this paper. In brief, the knowledge base of an agent a_1, represented as a CG system, can be symbolized as $KB_1 = <T_1,I_1,C_1,H_1,A_1>$. Agent a_1 will search KB_1 each time it decides to answer some query. Similarly, the query q emanates from an agent a_2 whose knowledge base KB_2 may be symbolized as $<T_2,I_2,C_2,H_2,A_2>$, with q being a graph that belongs to the universe of discourse of agent a_2. Figure 1 sketches a situation where CG-based agents would be available on the Web.

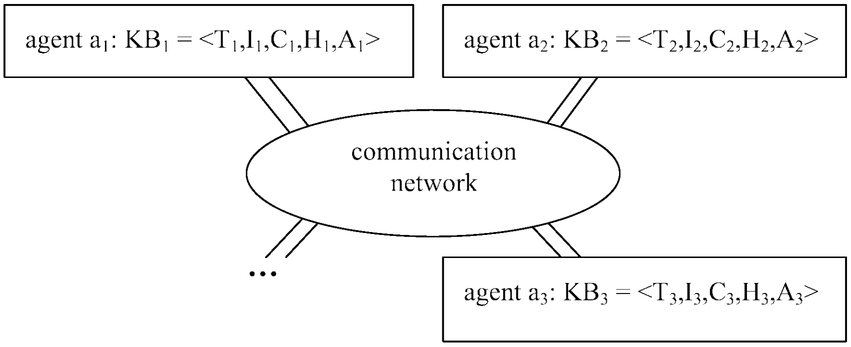
agent a_1: $KB_1 = <T_1,I_1,C_1,H_1,A_1>$
agent a_2: $KB_2 = <T_2,I_2,C_2,H_2,A_2>$
communication network
...
agent a_3: $KB_3 = <T_3,I_3,C_3,H_3,A_3>$

Fig. 1. The backbone of the knowledge Web

For any asserted conceptual graph g, and for $KB_i = <T_i,I_i,C_i,H_i,A_i>$, let us define functions: a) $type_i : A_i \rightarrow P(T_i)$ as the set of partially ordered types in T_i used in g, along with all generalizations and specializations of all types used in g (according to T_i)[3,4], b) $ref_i : A_i \rightarrow P(I_i)$ as the set of referents in I_i (constants and variables) used in g, c) $conf_i : A_i \rightarrow C'$, where C' is the subrelation of C_i defined over $type_i(g) \times ref_i(g)$ only, and d) $cons_i : A_i \rightarrow P(H_i)$ as a set of constraints in H_i to which g conforms, written $cons_i(g)::g$. Hopefully, $H_i::g\ \forall g \in A_i$, if KB_i is to be *consistent.* With these definitions, we can define the *context* of any graph g with regard to KB_i as: $<type_i(g), ref_i(g), conf_i(g), cons_i(g)>$, whether $g \in A_i$ or not.

[2] For a formal definition of a canon, please refer to [4].

[3] Here we use the notation P(S) to denote the partition set of any set S.

[4] We also assume that the $type_i$ function is order preserving, that is, the partial order of generality/specificity in T_i is preserved for all elements of $P(T_i)$.

In order to interpret a query q emanating from agent a_2, a_1 will need to be provided with the context of q with regard to KB_2 (where it originated): <$type_2(q)$, $ref_2(q)$, $conf_2(q)$, $cons_2(q)$>, and will compute the context of q with regard to KB_1: <$type_1(q)$, $ref_1(q)$, $conf_1(q)$, $cons_1(q)$>. The comparison of these two contexts will determine the level of interoperability between these two agents with regard to answering query q. This minimalist information interchange approach in which interoperability between agents intervene, is sketched in Figure 2 below and is the subject of the next two sections.

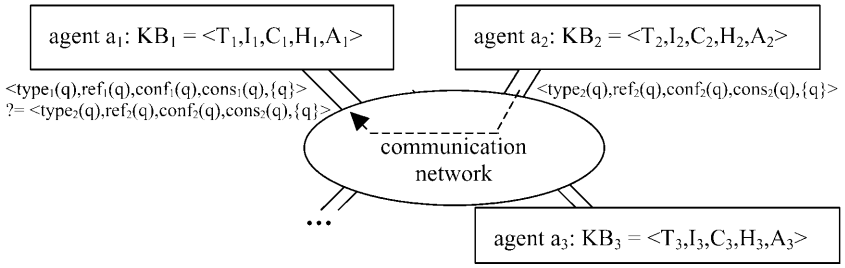
agent a_1: $KB_1 = <T_1,I_1,C_1,H_1,A_1>$
agent a_2: $KB_2 = <T_2,I_2,C_2,H_2,A_2>$
$<type_1(q),ref_1(q),conf_1(q),cons_1(q),\{q\}>$
$?= <type_2(q),ref_2(q),conf_2(q),cons_2(q),\{q\}>$
$<type_2(q),ref_2(q),conf_2(q),cons_2(q),\{q\}>$
communication network
...
agent a_3: $KB_3 = <T_3,I_3,C_3,H_3,A_3>$

Fig. 2. Piece-by-piece communication between agents

3 Computing Interoperability between CG-Based Systems

As presented bye Sowa in [13], the interoperability between communicating systems is much more than agreeing on some common representation standard, but goes deeper into the semantic representation of knowledge. In terms of CG-based systems, interoperability between two knowledge sources a_1 and a_2 depends on the compatibility between their individual canon. Compatibility may be partial, and thus, that is why we aim at maximizing its chances of success by computing it for every new query sent from a_2 to a_1, therefore between a context and a canon[5]. Compatibility between canons or contexts renders four types of compatibility assessments, one for each component of a context (or canon).

3.1 Compatibility of Types

In CG based systems, T_1 and T_2 represent partially ordered vocabularies. In the literature, the term *ontologies* is often used. Whether a_1 and a_2 are able to negotiate, use or infer the same (or partial) ontology is a rather difficult subject since it refers directly to the semantics of an application domain. Computing the compatibility between T_1 and T_2 is in itself a large endeavor since they may partially overlap. The smallest overlap of interest for the task at hand, i.e., to answer query q, is the one

[5] Of course a high volume of communication between a_1 and a_2 would entail computing their compatibility between their entire canons.

between $type_2(q)$ and T_1. Hopefully, if $type_2(q) = type_1(q)$, then at least all the types[6] in q are covered by an equivalent term in T_1. The only worry is with the semantics of these terms. Under a *single name assumption*[7] over $T^* = T_1 \cup T_2$, the types in q can be interpreted by a_1 without loss of meaning. To ensure that this is the case however, this assumption implies that some type compatibility resolution mechanism was carried out at some earlier stage. For instance, using predetermined (shared) ontologies (like WordNet [14]) as a basis for term selection between communicating agents makes that assumption. We leave to others this work on common ontology building; we believe that some imperatives of the market place, especially for business and government related applications, will force individual ontologies to be built and to be made available to their target users (probably at low cost). In what follows we assume that such a library of ontologies is commonly available, or that ontology mapping techniques are available within each agent to determine its level of compatibility with regard to the interpretation of some (partially) foreign vocabulary. We do not wish to address that very important issue for now, and leave other researchers tackle it; we chose to focus the bulk of our work on the other aspects of compatibility computation between CG-based agents.

3.2 Compatibility between Object Sets

The set of referents $ref_2(q)$ contains both constants and variables. In order to answer query q, a_1 must know all *constant* objects of $ref_2(q)$; so all constant objects in $ref_2(q)$ must appear in $ref_1(q)$. If not, then I_1 (and therefore $ref_1(q)$) must be extended to include these objects. Naturally one must make sure that no object ends up being named by two different constants; thus we assume here again a *single name assumption* for objects. This may entail the need for a (duplicate name) conflict resolution mechanism between a_1 (I_1) and a_2 (I_2) as I_1 is extended.

And for any variable v in $ref_2(q)$, I_1 must be extended in order to add a *new* variable v' to I_1 and to associate it with v (and otherwise avoid an early and not necessarily appropriate binding of variables[8]). In summary, compatibility resolution between object sets entails that in the end $ref_2(q) = ref_1(q)$, and that it may be necessary for a_1 to extend I_1 in order to reach that condition.

6 The reader should note that T includes all types, concept and relation types, and that the extensional semantics of the CG notation that we use is given in [4], where this is stated in a formal way.

7 The single name assumption implies that there exists a function *name* : $O \rightarrow L$, where O is the set of all objects needed either at the data or meta level to describe the application domain, and L is the set of labels (terms) used by the representation language (syntactical constructs) to refer to them.

8 Here the later binding operation is seen as external to the compatibility computation process, since it may vary greatly according to the nature or context of the application domain.

3.3 Compatibility between *Object to Type* Assignments

At this stage, $type_2(q) = type_1(q)$ (or there is a satisfactory mapping from $type_2(q)$ into $type_1(q)$), and $ref_2(q) = ref_1(q)$. So the objects of q are known and the vocabulary used to characterize them can be interpreted by a_1. However, it may be the case that agents a_1 and a_2 do not share the same viewpoints on the world, and some object in $ref_2(q)$ may not be typed the same way in $conf_1(q)$ as in $conf_2(q)$, which can be detected *automatically*. Let us define $t_1 \in T_1$ such that object i in $ref_2(q)$ conforms to it, written t_1::i, and such that there is no specialization t of t_1 in T_1 such that t::i. Then type t_1 is said to be the *maximally specific characterization* of i (in T_1). And let t_2 be the type associated with i in q. Under a normal form representation [7], q contains only one concept representing i, and therefore, we know that t_2 is unique. Provided that $type_2(q) = type_1(q)$, we have one of the following cases:

$t_1 \leq t_2$ and the concept representing i in q, [t_2:i], can be interpreted as such by a_1,

$t_1 > t_2$ and the concept representing i must be modified in order for a_1 to produce some answer to q (i.e., concept [t_2:i] in q must be changed to [t_1:i], if possible[9]),

t_1 and t_2 are not comparable and then concept [t_2:i] in q is changed to [t_3:i] where t_3 is the maximally specific generalization of both t_1 and t_2, if possible[10].

Of course, the generalization step(s) that may be required to answer query q may produce a query graph whose answer would include more data than originally expected. Data filtering may be required in order: a) to avoid providing unnecessary data to a_2, and b) to protect secure data of a_1 from being accidentally accessed by the query. Sections 3.4 and 4 below explain how the constraint mechanism described in [2] can be used to filter out data that either a_1 does not want to reveal, or a_2 does not wish to get.

3.4 Compatibility between Query and Constraints

There is usually little need for a_1 to know whether $cons_2(q)$::q or not, so $cons_2(q)$ could be set to the empty set (since $\varnothing$::g holds, by definition, for any conceptual graph g). However, in a query answering setting as we described above, it may be interesting for a_2 to identify a set of constraints that the answers to q should be compliant with. Therefore, $cons_2(q)$ may be used for that purpose. This is a way to filter out answers to q that are not desired by a_2. Provided that a_2 is willing to give

9 If not possible, that means that this generalization violates some constraint on the relations that may be attached to [t_2:i], and that therefore the relations that can not be attached to [t_2:i] anymore (since it must be rewritten as [t_1:i]) must be detached from it. As a result, the resulting query q' may be a disconnected graph, each connected component being treated by a_1 as an independent query graph.

10 Same as footnote 9 above. The reader will notice that the $\top$ and $\bot$ elements of T_i are always part of any $type_i$ set, and that therefore $type_i$ forms a lattice structure.

that information to a_1, the amount of information transferred from a_1 to a_2 would be less if a_1 applied this filter onto its generated output, and less processing would be required by a_2 in order to answer the query[11] than if a_2 filtered out the resulting set of graphs itself. Let us define q* as the set of answers to q (from a_1). Then in that case we would require that ∀q' in q*, $cons_2(q)$::q' holds. Naturally, in order for a_1 to interpret $cons_2(q)$, $type_2(q)$ must include all types (and their generalizations and specializations) found in all graphs of $cons_2(q)$, and $ref_1(q)$ must be extended in the same way as explained in Section 3.2 above, but using all constants and variables found in all constraint graphs of $cons_2(q)$. Section 4.1 briefly presents how semantic constraints can be represented under the CG formalism.

Also, in order to avoid giving access to private data when answering a query q, all graphs that instantiate query q and that encode private data should not be part of q*; some filter mechanism should be used in order to discard these graphs. Section 4.2 below shows how to represent such a filter mechanism as a set of constraints H describing what graphs could exist with regard to some outside view of the data, and what other graphs could not. With that framework, ∀q' in q*, we have that (H ∪ $cons_2(q)$)::q' must hold. So all graphs in q* will be computed in light of (H ∪ $cons_2(q)$), an *extended* set of constraints. That way a_1 will not give access to protected data; and a_2 will not receive unwanted information. Section 4 describes how a filter mechanism can be represented as a set of constraints H to satisfy.

4 Compatibility over Sets of Constraints

First, Section 4.1 reminds the reader of the representation framework introduced in [2] to model semantic constraints under the CG formalism. Then, Section 4.2 formulates the filter problem as a constraint satisfaction problem, thus allowing the use of constraints to implement a filter mechanism over queries.

4.1 Semantic Constraints under the CG Formalism

Different proposals exist in the CG literature to represent semantic constraints [2,8,9,10]. To our opinion, the most complete proposals in terms of their coverage are [2,9], and we feel that their use in the representation of semantic filters would probably be equivalent. Being directly involved in [2], we chose to use that framework to further describe how it could be used to implement filters (Section 4.2). Therefore this section summarizes in a nutshell our previous work on semantic constraints.

In [2] we presented two classes of constraints: *domain* and *topological* constraints. Domain constraints are those that restrict the set of values that can instantiate a

[11] And this is particularly interesting for applications which dispatch many queries at once and which may therefore receive many answer sets at the same time, like for example, broadcast applications deployed on distributed databases and whose primary purpose is to manage distributed database queries.

variable in a generic concept of some graph; topological constraints restrict the set of graphs that can be asserted. In what follows we concentrate on the latter though both are needed to fully describe all semantic constraints normally found in database literature [11].

As the reader probably recalls, the set of all asserted graphs in a CG system forms a generalization hierarchy. In effect, for any two graphs g_1 and g_2, either g_1 is more specific than g_2, written $g_1 <_g g_2$, or g_1 is more general than g_2, written $g_1 >_g g_2$, or both are equivalent, written $g_1 =_g g_2$, or they are incomparable, written $g_1 \neq_g g_2$. Let us define G the set of all *asserted* graphs A in a CG system together with the partial order relation $<_g$ that is inferable between all pairs of graphs in A by the application of the projection operator π [3]; so $G = <A,>_g>$. Let us define A* as the set of all *derivable* graphs from the canon of the system. And let us define G* as $<A^*,>_g>$. Clearly $G \subseteq G^*$ since the graphs of A are all derivable from the canon and must also appear in A*, and since the partial order relation between all pairs of graphs does not change whether the graphs are in A or A*.

In [2] we defined a semantic constraint c as a subhierarchy $G_c \subseteq G^*$, where all graphs in G_c, though derivable from the canon, should not be asserted in A in order to avoid violating constraint c. G_c can be represented in a very compact way; constraint c identifies the most general graph g in G_c (which we defined as unique) that *should not* appear in A even though it is derivable from the canon. Figure 3 illustrates a constraint that states that: *"there is no employee that manages a project to which s/he is assigned"*.

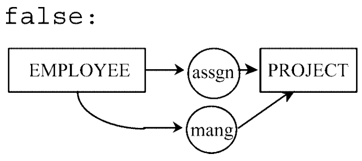

Fig. 3. A conceptual graph g used as a constraint

So g should never be asserted in A, and neither should any of its specializations. G_c is represented by its most general graph g and implicitly by all of g's specializations. The set of all semantic constraints associated with a domain is called H and is part of the canon of the system since it restricts the subsequent assertion/derivation of graphs. Asserting any graph g' in A should be done with respect to H, that is g' will be asserted into A if it does not fall within G_c, for any c in H. Consequently $G_c \subseteq G^*$, but $A \cap$ (the graphs in G_c) = $\emptyset$ for all constraints c in H. So one could see that $\cup_{c\in H} G_c$ as an overlay defined over G*, determining invalid assertion subspaces of the universe of discourse.

Our work in [2] presents not only constraints, but *constraints with exceptions*. In that case G_c does not include all specializations of g, its most general graph, but some subhierarchies of G_c may be excluded from it by defining exceptions to c. We will not present that part of our work here but the fundamentals of what we presented in this section remain true in that case as well. For what follows the reader must only remember that a set of semantic constraints H defines an overlay over G* the set of all derivable graphs from the canon, which constrains which graphs can be asserted into A. Whether H is composed of *partial* or *complete* G_Cs, representing constraints *with* or *without* exceptions, is not relevant for our current argumentation[12].

1.2 Filters as Sets of Constraints

Now that we can see H as an overlay over G*, the set of partially ordered conceptual graphs derivable from the canon of a CG system, it is straightforward to define its use in the implementation of semantic filters.

As said in Section 4.1 above, G_C prevents assertions to be made in a specific subhierarchy of G*, leaving the corresponding part of G empty. By doing that, all graphs in A conform to H, and the system is said to be consistent with regard to its constraint set. If some agent wishes to block access to part of its data set, it could consider these parts (in G) as being empty when seen *from the outside world* (when trying to answer a query). Therefore, an agent could define some overlay H' over G*, then called *semantic filter*, in the same way its constraint set H is defined, but which would identify subspaces of G* that should be considered empty when answering outside queries. For instance, if an agent does not wish to let the outside world know whether some project manager works on projects to which s/he is assigned, graph g of Figure 3 could be part of H', defining a subhierarchy of G*, called G_g, that identifies graphs of A that should not be part of any answer set q*. That way, when answering a related query q, though there may be graphs in A that would normally instantiate the query and would be part of q*, the provided answer to the outside world would be Q $= q^* \setminus \{g' \mid g' \in G_g \; \forall g \in H'\}$, where q* is the set of all graphs in A which embed some projection of q[13].

As the computation of Q above shows, our model deletes from the answer set, all graphs that contain part of some private data, instead of cutting out the parts that should not be seen by the outside world. This choice was made in order to avoid data reconstruction from external agents that would send sequences of overlapping queries, and that, with some inference mechanism, could guess with a high probability, what the missing pieces could be.

In a world where communication with the outside world allows for various security level clearances, it would be desirable to have semantic filters that are custom made. That is, depending on the origin of a query, a different filter would be used, providing different views over the same data set. These views are in fact *interfaces* between communicating agents. Figure 4 shows the schema of Figure 1 updated accordingly. Such an architecture is directly related to the modeling of the different agents with which an agent interacts, and will be discussed in a forth-coming paper.

[12] In [4] we present the extensional semantics of the CG notation, including that of semantic constraints. The interested reader should refer to it.

[13] Of course this filtering out of graphs from q* need not be done after q* is computed, but can be embedded in the evaluation of q* itself. In fact, by using the elements of H' to determine the subspaces where the answer to q could lie, a gain in performance could be achieved as noted in [12]. The precise evaluation of this gain is yet to be done.

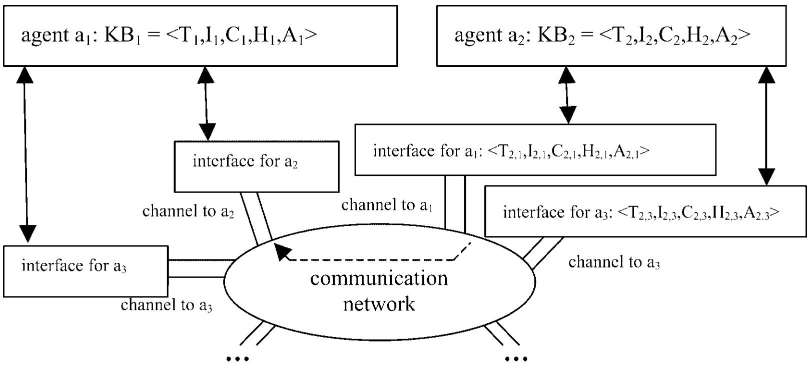
agent a1: KB1 = <T1,I1,C1,H1,A1>
agent a2: KB2 = <T2,I2,C2,H2,A2>
interface for a2
interface for a1: <T2,1,I2,1,C2,1,H2,1,A2,1>
interface for a3: <T2,3,I2,3,C2,3,H2,3,A2,3>
interface for a3
channel to a2
channel to a1
channel to a3
channel to a3
communication network
…
…

Fig. 4. The communication of agents through *external views* over available data sets

5 Conclusion and Future Directions

As the semantic Web will eventually offer structured knowledge within Web documents, the next stage of development for the Web will be to offer knowledge-oriented services altogether, giving birth to the *knowledge Web*. Therefore, there will be a need for knowledge providers (and eventually for knowledge brokers) available on-line, answering the various requests coming to them from every part of the Web. Agent technology will probably be best suited to implement such knowledge providers. We see the CG formalism as the representation language for describing these agents because it is flexible, very expressive, formally defined, and easy to learn and use (mainly because of its graphical nature and its closeness to the UML and ER modeling languages).

Toward that goal, this paper discusses some issues regarding the interoperability of CG-based systems in the light of a single problem: the answering of a query q by an agent a_1, sent by some other agent a_2. It defines the notion of a *context of a graph*, which is the subset of a canon upon which the graph is based. It proposes to establish a mapping between two contexts of the same graph, one being the context of the graph in the source domain, the other one, the context of the graph in the target domain, and shows how agent a_1 can detect what are its shortcomings with regard to the interpretation of query q, in a *totally automatic manner*. Based on this information, it is assumed that some resolution mechanism could be triggered if needed.

As the main trend in the literature today with regard to this resolution mechanism is centered around the establishment of a common ontology, we chose to rather discuss the handling of semantic constraints in the process of answering query q. For that purpose, this paper proposes to use a simple overlay mechanism, based on the representation of semantic constraints as presented in [2], in order to implement *semantic filters* required to control, on a semantic level, the information that is exchanged between communicating agents. It is easy to see that *communication interfaces* between agents could provide for different security clearance levels.

Now that we have established what is required of CG-based systems in order for them to be interoperable, and that we have proposed a model to compute the interoperability level between communicating agents, we will focus our attention on providing algorithms that will help knowledge engineers negotiate a common context for a query graph and its answer set. We foresee that abduction and probabilistic reasoning techniques will be part of the solution, as tentative mappings between contexts will be proposed and eventually revised as more graphs are exchanged between agents. By defining a model of interoperability computation between CG-based communicating agents, this paper laid down the ground work needed to bridge the gap between various knowledge centered applications (knowledge providers, brokers, users) on the Web, which is absolutely required to eventually implement the knowledge Web.

References

1. Edwards, K.W., (1999). *Core jini*. Prentice Hall.
2. Mineau, G.W. & Missaoui, R., (1997). The Representation of Semantic Constraints in CG Systems. Conceptual Structures: Lecture Notes in Artificial Intelligence, vol. 1257. Springer-Verlag. 138-152.
3. Sowa, J. F., (1984). Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley.
4. Mineau, G.W., (2000). The Extensional Semantics of the Conceptual Graph Formalism. Lecture Notes in AI, vol. 1867. Springer-Verlag. 221-234.
5. Mineau, G.W., (1999). Constraints on Processes: Essential Elements for the Validation and Execution of Processes. Lecture Notes in Artificial Intelligence, vol. 1640. Springer-Verlag. 66-82.
6. Mineau, G.W. & Gerbé, O., (1997). Contexts: A Formal Definition of Worlds of Assertions. Lecture Notes in Artificial Intelligence, vol. 1257. Springer-Verlag. 80-94.
7. Chein, M. & Mugnier, M.L., (1993). Specialization: Where Do the Difficulties Occur? Lecture Notes in Artificial Intelligence, vol. 754. Springer-Verlag. 229-238.
8. Dibie-Barthélemy, J., Haemmerlé, O. & Loiseau, S., (1998). Refinement of Conceptual Graphs. Lecture Notes in Artificial Intelligence, vol. 2120. Springer-Verlag. 216-230.
9. Dibie, J., (1998). A Semantic Validation of Conceptual Graphs. Lecture Notes in Artificial Intelligence, vol. 1453. Springer-Verlag. 80-93.
10. Pfeiffer, H.D. & Hartley, R.T., (1992). Temporal, spatial, and constraint handling in the Conceptual Programming environment, CP. *Journal of Experimental & Theoretical Artificial Intelligence*; **4**(2). 167-183.
11. Elmasri, R. & Navathe, S.B., (1994). *Fundamentals of Database Systems*. 2nd edition. Benjamin Cummings.
12. Levinson, R. A. & Ellis, G., (1992). Multi-level hierarchical retrieval. *Knowledge Based Systems*, **5**(3). 233-244.
13. Sowa, J. F., (2002). *Negotiation Instead of Legislation*. Available at: www.jfsowa.com/talks/ negotiat.htm.
14. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D. & Miller, K.J., (1990). Introduction to WordNet: an on-line lexical database. Int. Journal of Lexicography, Vol.3, No 4, 235-244.