1 METAREASONING AGENTS FOR QUERY-ANSWERING SYSTEMS

Jonas Barklund
Pierangelo Dell'Acqua

Computing Science Dept.
Uppsala University
Box 311, S-751 05 Uppsala
Sweden

{jonas, pier}@csd.uu.se

Stefania Costantini
Gaetano A. Lanzarone

Dip. di Scienze dell'Informazione
Università degli Studi di Milano
via Comelico 39/41, Milano
Italy

{costanti,lanzaron}@dsi.unimi.it

Abstract: In this paper we claim, and demonstrate by means of a number of examples, that a formalism equipped with metalogic and reflection capabilities allows us to easily model (by means of the metalevel) and implement (by means of reflection) the desired degree of flexibility in a query-answering system interacting with a data/knowledge base. We also try to show that a main feature of building a really flexible system is the possibility of modeling metalogical agents, which are able to interact and to communicate in various different ways. This because it is useful to view a query-answering system as a collection of agents including, in particular, some external (typically human) agents, in order to make a system behave more rationally and flexibly towards users.

1.1 INTRODUCTION

The perception of a user about "flexibility" of interaction with a computer system, and in particular with a data/knowledge base (DKB), can be based on several aspects, including the following three:

Syntactic flexibility: the system should be able to answer a question even in case the syntax of the question does not coincide with the internal representation; also, the system should be able to adapt the syntax to the needs/skills of the particular user. Syntactic flexibility is important in at least two situations. (i) Reuse of an existing DKB in a different context, and/or merge of different

DKB's. The resulting DKB should answer the former queries, as well as new, more complex ones. (ii) Interaction with users that either do not know or are not allowed to see the actual structure of the DKB, and then ask queries on the basis of their own "view" of the DKB.

Reasoning flexibility: the system should be able to employ non-classical or non-deductive forms of reasoning such as analogy, induction, etc. The user may possibly be asked to validate the conclusions.

Knowledge-level flexibility: according to a classification of the users and/or of the questions, the system should be able to use different portions of knowledge and/or different deduction strategies.

In this paper we claim, and demonstrate by means of a number of examples, that a formalism equipped with metalogic and reflection capabilities (introduced in Section 1.2) allows us to easily model (by means of the metalevel) and implement (by means of reflection) the desired degree of flexibility. In Section 1.3 in fact we show how metalogical features can be useful for: (i) easily implementing many features of flexible query-answering systems; (ii) improving the existing DKB's w.r.t. their flexibility in query answering, by defining a suitable interface toward the user. An important advantage of metalogic procedures is that they can be developed and modified quite easily both by a system manager or by an (experienced) user. Then a user is, in principle, in the condition of "tuning" the behaviour of a DKB according to her own needs.

We also claim that a main feature for building complex and really flexible DKB's is the possibility of modeling agents, which are able to interact and to communicate in different ways. Agents incorporate in principle different reasoning capabilities and/or different knowledge and should exhibit a rational behaviour when interacting with each other and with humans. Agents should have metareasoning capabilities, i.e. they should be capable of reasoning about their own beliefs and inferences, as well as those of other agents. With "*interacting*" with each other we primarily mean agents asking questions and agents telling propositions, but it could also involve other acts, such as exchanging money for services. With "*rational behaviour*" in this context we mean that an agent should behave differently, according to its own model of the other agents involved in the interaction. In this paper, we first introduce metalogical reflective agents (Section 1.2), and then (Section 1.4) we try to demonstrate that modeling a DKB as a collection of metalogical agents that interact with other external (typically human) agents is useful for obtaining query-answering systems that behave more rationally and flexibly towards users.

Our argumentation will consist of a collection of examples of how to express different forms of interaction with a DKB by using a first-order logic language with metalogical extensions. In particular, we refer to the language Reflective Prolog [Costantini, Lanzarone, 1989, Costantini, Lanzarone, 1994]. We will try to show how the metalogical capabilities allow us to achieve various modalities of interaction, even though in principle they could be achieved using only object level language. (This is similar to the observation how programming in a high-level programming language makes it possible to develop applications that in principle could have been written in assembly languages although that would not have been easily feasible in practice.)

1.2 A LOGIC LANGUAGE WITH METALOGICAL EXTENSIONS

In this section we introduce Reflective Prolog (*RP*) a metalogic programming and knowledge representation language. *RP* is an extension of the language of Horn clauses [Kowalski, 1974, Lloyd, 1987] to contain higher-order-like features, metalevel negation and multiple theories (*agents*). Since all these extensions rely upon the same kind of logical reflection [Feferman, 1962, Costantini, 1990], agents and metalevel features of the language can be naturally combined, returning greater expressive and problem-solving power.

Reflective Prolog

RP extends the language of Horn clauses to an enhanced language containing names of the expressions of the language itself. *RP* allows significant freedom in the choice of names: we only require that names of compound expressions be compositional. We assume as given the specification of a full *naming mechanism* which allows the representation of metaknowledge. The naming mechanism must obey certain requirements, but in principle may vary depending on the needs of a specific application domain. We can say that the language takes the naming mechanism as a parameter. In the following, the name of an expression α is conventionally indicated with $'\alpha$. (For sake of simplicity we use a different notation from [Barklund *& al*, 1994, Barklund *& al*, 1995].) If, for instance, we want to express properties (*metaknowledge*) of an expression of the object language (*knowledge*) such as $p(a, x)$, we have to employ a name of that expression, represented here as $'p('a, 'x)$, where $'x$ stands for the name of what will be the value of the variable x. We may for example express that p is a binary predicate symbol as: $binary\_pred('p)$. Notice that we have employed the name of p and not p itself since a predicate symbol cannot appear in a term position.

The language is that of Horn clauses except that terms are defined differently in order to include *names* that are intended to represent the expressions of the language itself. The alphabet of *RP* differs from the usual alphabet of Horn clauses by making a distinction between variables and *metavariables* and through the presence of *metaconstants*. Metavariables can only be substituted with names of expressions of *RP*, and metaconstants are intended as names for constants, function and predicate symbols of RP. If c is a constant, a function or a predicate symbol, then we write $'c$ as a convenient way to represent the metaconstant that names c. Similarly, if $'c$ is a metaconstant, then its name is $''c$, and so on. Furthermore, the alphabet of RP contains a denumerable set of constants, called *theory symbols*, and the predicate symbols *solve*, *solve_not*, *tell* and *told*.

Notation: lower-case characters (possibly indexed) are reserved for variables, and upper-case characters (possibly indexed) are reserved for metavariables. Words consisting of at least two characters are reserved for the remaining symbols of the alphabet. Thus x and y_3, for example, are used for variables, A and X_3 are used for metavariables, and *ap1* and *John* may be used for constant symbols.

RP allows the possibility of specifying *metaevaluation clauses* that are clauses defining the (unary) predicate symbols *solve* and *solve_not*. They both take as argument the name of an atom whose predicate symbol can neither be *solve* or *solve_not*, nor *tell* or *told*, that is, meta-metaevaluation is not permitted in the language. *Solve* allows us to extend the intended meaning of predicates, while *solve_not* restricts the intended meaning of predicates: an atom A is not a consequence of a RP program if $\mathit{solve\_not}('A)$ is.

Suppose, for example, that we want to express the fact that an object *obj* satisfies all the relations in a given *class*. If we want to formalise that statement by object level clauses, then for every predicate *pred* in *class* we have to write the clause $\mathit{pred}(\mathit{obj})$. Instead, we may formalise our statement at the metaevaluation level as

$$\mathit{solve}(P('\mathit{obj})) \leftarrow \mathit{belongs\_to}(P, \mathit{class})$$

where P is a metavariable ranging over the names of predicate symbols.

Clauses for *solve_not* specify when predicates do not hold. Consider for example the object level clause stating that two individuals, x and y, are brothers if they have a parent in common:

$$\mathit{brother}(x, y) \leftarrow \mathit{parent}(p, x), \mathit{parent}(p, y)$$

An undesired consequence of this clause is that $\forall x.\mathit{brother}(x, x)$ holds, that is, everybody is brother of himself. A possible way of dealing with such a situation is to state that the predicate *brother* is irreflexive, that is, $\forall x.\neg \mathit{brother}(x, x)$. This may be expressed in RP as:

$$\begin{array}{l} \mathit{irreflexive}('\mathit{brother}) \\ \mathit{solve\_not}(P(X, X)) \leftarrow \mathit{irreflexive}(P) \end{array}$$

The metaevaluation clause above states that the couple of individuals denoted by (X, X) is not in the relation denoted by P if P is asserted to be irreflexive. That metaevaluation clause applies to every predicate declared irreflexive.

Agents and communication among agents

RP has been extended to incorporate communication-based reasoning by [Costantini *& al*, 1992]. The main idea of the approach is to represent agents and communication acts by means of theories and reflection principles, respectively. Thus, theories formalise knowledge of agents, while reflection principles characterise possible kinds of interaction among agents. To indicate that an atom A belongs to a theory ω we label A with ω, that is, we write $\omega{:}A$. Given a clause $A{\leftarrow}B_1, \ldots, B_n$, we write $\omega : A{\leftarrow}B_1, \ldots, B_n$ as an abbreviation for the labelled clause $\omega{:}A \leftarrow \omega{:}B_1, \ldots, \omega{:}B_n$.

An *agent* is composed of a theory (called the *associated theory* of the agent) and implicitly by the inference mechanism of the language. Thus, declaratively an agent is denoted by the declarative semantics of the associated theory, and procedurally by the closure of the theory itself under resolution.

Communication acts are formalized by means of the (binary) predicate symbols *tell* and *told*. They both take as first argument the name of a theory symbol and as second argument the name of an expression of the language. Let ω_1 and ω_2 be theory symbols and A an atom. The intended meaning of $\omega_1{:}tell('\omega_2, 'A)$ is: the agent ω_1 tells agent ω_2 that A, and of $\omega_2{:}told('\omega_1, 'A)$ is: ω_2 is told by ω_1 that A. These two predicates are intended to model the simplest and most neutral form of communication among agents, with no implication about provability (or truth, or whatever) of what is communicated, and no commitment about how much of its information an agent communicates and to whom. An agent ω_1 may communicate to another agent ω_2 everything it can derive (in its associated theory), or only part of what it can derive, or it may even lie, that is, it communicates something it cannot derive.

The intended connection between *tell* and *told* is that an agent ω may use (by means of *told*) only the information that another agent has explicitly addressed to it (by means of *tell*). Thus, an agent can specify, by means of clauses defining the predicate *tell*, the modalities of interaction with the other agents.

What use ω makes of this information is entirely up to ω itself. Thus, the way an agent communicates with others is not fixed in the language. Rather, it is possible to define in a program different behaviours for different agents, and/or different behaviours of one agent in different situations as the examples of Section 1.3 show. (More elaborate examples of the use of agents in *RP* can be found in [Costantini *& al*, 1992].)

The peculiarity of this approach to theories is that the primitive introduced in the language are not aimed at structuring programs, are not intended as schemas of composition of theories, and are not concerned with logical/ontological modelling of concepts like belief or knowledge. They are simple communication means among theories, on top of which more purposeoriented mechanisms may be built. In the next section, for example, we provide agents with metalevel reasoning capabilities.

Logical reflection

Both communication acts and the metalevel features of the language are formalised via a form of logical reflection.

Communication acts are formalized by means of (inter-theory) reflection axioms based on the predicate symbols *tell* and *told*:

$$\omega_1{:}told({}'\omega_2, {}'A) \leftarrow \omega_2{:}tell({}'\omega_1, {}'A)$$

Its intuitive meaning is that every time an atom of the form $tell({}'\omega_1, {}'A)$ can be derived from a theory ω_2 (which means that agent ω_2 wants to communicate proposition A to agent ω_1), the atom $told({}'\omega_2, {}'A)$ is consequently derived in the theory ω_1 (which means that proposition A becomes available to agent ω_1).

The intended connection between the object level and the metalevel of *RP* is obtained by means of the following (inter-level) reflection axiom:

$$\omega{:}A \leftarrow \omega{:}solve({}'A)$$

It asserts that whenever an atom of the form $solve({}'A)$ is provable at the meta-evaluation level of a theory ω, then A is provable at the object level of ω itself.

Reflection axioms make the extensions of the language effective both semantically and procedurally [Costantini, Lanzarone, 1989].

Semantically, this has been achieved by introducing the notion of *reflective model* and *reflective logical consequence* [Costantini, 1990]. Reflective models are models in the usual sense. For a logic program P without *solve_not* clauses the model intersection property still holds, and therefore there exists the least reflective model RM_P of P. It can be characterised as the least fixed point of a mapping T_P similar to the usual one [Lloyd, 1987], except that it produces both $\omega{:}A$ and $\omega{:}solve({}'A)$ whenever one of them can be derived in a theory ω. We have that $RM_P = lfp(T_P) = T_P \uparrow \omega$ [Costantini, 1990].

For a programs P with metalevel negation (*solve_not* clauses), among the minimal models of P, we choose the one that never entails $\omega{:}A$ if it entails $\omega{:}solve\_not({}'A)$. It can be obtained [Costantini, Lanzarone, 1994] as the iterated least fixed point IRM_P of a mapping $T_{J(I)}$. $T_{J(I)}$ is based on T_P, where an atom $\omega{:}A$ cannot be included in $T_{J(I)}$ if $solve\_not({}'A)$ belongs to J.

Procedural semantics of *RP* programs is based on an extension of SLD-resolution that allows both inter-theory communication and the shifting between the object level and the metalevel of a theory. Informally, the extended SLD-resolution is able to use labelled clauses with conclusion $\omega_1{:}tell({}'\omega_2, {}'A)$ to resolve a query $\omega_2{:}told({}'\omega_1, {}'A)$, labelled clauses with conclusion $\omega{:}solve({}'A)$ to resolve a query $\omega{:}A$, and vice versa clauses with conclusion $\omega{:}A$ to resolve a query $\omega{:}solve('A)$.

Procedural semantics for RP programs without metalevel negation is called RSLD-resolution (sound and complete with respect to RM_P). Let G be a query of the form ?- $\omega_1{:}A_1, \ldots, \omega_n{:}A_n$ and let $\omega_m{:}A_m$ be the selected labelled atom in G. Then ?- $(\omega_1{:}A_1, \ldots, \omega_{m-1}{:}A_{m-1}, \omega_m{:}B_1, \ldots, \omega_m{:}B_r, \omega_{m+1}{:}A_{m+1}, \omega_n{:}A_n)\theta$ is derived from G and labelled clause C using substitution θ if and only if one of the following conditions holds:

1. A_m is distinct from $told(\_,\_)$
   C is $\omega_m : A \leftarrow B_1, \ldots, B_r$
   θ is an mgu of A_m and A

2. A_m is $solve(\_)$
   A is distinct from $solve(\_)$, $tell(\_,\_)$ and $told(\_,\_)$
   C is $\omega_m : A \leftarrow B_1, \ldots, B_r$
   θ is an mgu of A_m and $solve('A)$

3. A_m is distinct from $solve(\_)$, $tell(\_,\_)$ and $told(\_,\_)$
   C is $\omega_m : solve('A) \leftarrow B_1, \ldots, B_r$
   θ is an mgu of $solve('A_m)$ and $solve('A)$

4. A_m is $told('\omega,\_)$
   C is $\omega : tell('\omega_m, 'A) \leftarrow B_1, \ldots, B_r$
   θ is an mgu of A_m and $told('\omega, 'A)$

Procedural semantics for RP programs with metalevel negation is based on RSLD-resolution (sound and complete with respect to IRM_P [Costantini, Lanzarone, 1992]). In particular, if a query G succeeds with computed answer θ (by means of RSLD-resolution), then $solve\_not('G\theta)$ is attempted:

- if $solve\_not('G\theta)$ succeeds, then G is forced to fail;

- otherwise, if $solve\_not('G\theta)$ fails, then the success of G is confirmed.

1.3 FLEXIBLE QUERY ANSWERING IN DKB'S: THE ROLE OF METALOGIC AND REFLECTION

In order to make information systems available to a more general public, flexible query answering systems need to be developed in such a way that they respond "intelligently" to the end-user. In the last decade of research in expert databases, for instance, considerable effort has been put in developing user-friendly interfaces. The main idea of these approaches consists in providing some flexibility to the user by allowing queries with less rigidity. [Loo *& al*, 1994], for example, investigate the possibility to allow *inadequate queries.* They include:

- *vague queries*: the user does not have a clear notion of what information he wants to get from the information system; a vague or ambiguous query may be constructed as an initial attempt to get a clue of what information is available.
- *imprecise queries*: the language of the query needs not conform with the syntax of the query language, or a query may contain syntax errors.
- *incomplete queries*: not enough information is specified in the query, or the data available in the database may be relevant but may not provide the exact values as requested.

Along the ideas outlined by Loo we show by several examples that a good deal of syntactic flexibility in query-answering can be easily achieved if the language is provided with metalevel capabilities.

1) Changes in the representation of relations
Consider the following taxonomy:

$$\begin{array}{ll} KB: & human('man) \\ & animal('human) \\ & man('John) \end{array}$$

Assume that a user may want to ask queries in a different syntax, like for instance:

$$?\text{-}\ KB{:}is\_a('man, 'human)$$

This is allowed by the following metaevaluation clause which transforms one representation into the other.

$$KB: solve('is\_a(X, 'Y)) \leftarrow solve(Y(X))$$

2) Definition and use of intensional properties of relations
By means of metaevaluation clauses it is possible to define properties of relations. We may for instance introduce the concept of symmetry and irreflexivity in a theory KB as:

$$\begin{array}{ll} KB: & solve(X(Y,Z)) \leftarrow symmetric(X), solve(X(Z,Y)) \\ & solve(X(Y,Z)) \leftarrow transitive(X), solve(X(Y,W)), solve(X(W,Z)) \\ & solve\_not(X(Y,Y)) \leftarrow irreflexive(X) \end{array}$$

The first clause states that the objects whose names are Y and Z are in the relation whose name is X, provided that the relation denoted by X is asserted to be symmetric and that the objects denoted by Z and Y are in the relation denoted by X. The third clause states that the couple of objects denoted by (Y, Y) is not in the relation denoted by X, if X is asserted to be irreflexive. Thus, if *KB* also contains the clauses:

$$\begin{array}{ll} KB: & brother(John, Bob) \\ & symmetric('brother) \\ & transitive('brother) \\ & irreflexive('brother) \end{array}$$

then the system answers the following queries as:

$$\begin{array}{ll} ?\text{-} & KB{:}brother(Bob, John) \\ & yes \\ ?\text{-} & KB{:}brother(Bob, Bob) \\ & no \end{array}$$

Notice that if the metaevaluation clause

$$solve\_not(X(Y,Y)) \leftarrow irreflexive(X)$$

would not have been present in *KB*, then the system would have answered positively the query *?- KB:brother*(Bob, Bob).

3) Use of dictionary/thesaurus/encyclopedia

Flexibility in query answering can be increased if we allow queries which do not exactly match the data in the database, but rather the query can be reinterpreted in the database itself with respect to some other source of information: may it be a dictionary, a list of synonyms or antonyms (3a,3b), an encyclopedia (3c), etc.

$$\begin{array}{lll} \text{3a)} & KB{:} & solve(X(Y,Z)) \leftarrow symmetric(X), solve(X(Z,Y)) \\ & & solve(X(Y)) \leftarrow synonym(X, X_1), solve(X_1(Y)) \\ & & solve(X(Y)) \leftarrow synonym(Y, Y_1), solve(X(Y_1)) \\ & & symmetric('synonym) \\ & & synonym('amusing, 'entertaining) \\ & & synonym('puppet, 'marionette) \\ & & amusing(marionette) \end{array}$$

$$\begin{array}{ll} ?\text{-} & KB{:}entertaining(puppet) \\ & yes \end{array}$$

3b) *KB:* $solve(X(Y,Z)) \leftarrow symmetric(X), solve(X(Z,Y))$
$solve\_not(X(Y)) \leftarrow antonym(X,X_1), solve(X_1(Y))$
$symmetric('antonym)$
$antonym('married, 'single)$
$married(Bob)$

?- *KB:*$single(Bob)$
no

3c) *KB:* $solve(X(Y,Z)) \leftarrow symmetric(X), solve(X(Z,Y))$
$solve(X(Y,Z)) \leftarrow same\_pair((Y,Z),(Y_1,Z_1)), solve(X(Y_1,Z_1))$
$same\_pair((X,Y),(X_1,Y)) \leftarrow same(X,X_1)$
$same\_pair((X,Y),(X,Y_1)) \leftarrow same(Y,Y_1)$
$symmetric('same)$
$same('Boston, 'capital('Massachusetts))$
$population(Boston, 2million)$

?- *KB:*$population(capital(Massachusetts), 2million)$
yes

4) Relaxation by taxonomies

[Andreasen, Christiansen, 1996] propose a flexible interpretation of queries by means of predicate similarities. Suppose that a predicate q is defined similar to a predicate p, then the system may include in the answer of an empty extension of p the extension of q. Similarity can be defined, for example, relative to a taxonomy such that two sub-concepts are considered similar in case they have the same super-concept. This kind of taxonomy based query-weakening may be formalised as:

KB: $solve(X) \leftarrow relaxation(X,Y), solve(Y)$
$relaxation(P(X),Q(X)) \leftarrow super\_concept(P,W), super\_concept(Q,W)$
$super\_concept('pizza\_kebab, 'pizza\_like\_food)$
$super\_concept('pizza\_pie, 'pizza\_like\_food)$
$pizza\_pie(calzone)$

?- *KB:*$pizza\_kebab(x)$
$x = calzone$

5) Retrieval of properties of individuals

Assume that a user, say Juliette, wants to retrieve some information about an individual, say Andrew, without knowing exactly what, or without knowing which kind of information is available in the database. Then Juliette would like to be able to ask vague queries of the form: "return all the properties of Andrew", "return only the properties of Andrew that satisfy these requirements", and so on.

In the proposed approach this kind of flexibility can be obtained by posing queries directly at the metaevaluation level. Suppose that the database *KB* contains the facts:

$$
\begin{array}{ll}
KB\text{:} & young\_man(Andrew) \\
 & student(Andrew) \\
 & handsome(Andrew) \\
 & rich(Andrew) \\
 & \ldots \\
 & worker(Bob) \\
 & poor(Bob) \\
 & \ldots
\end{array}
$$

and assume that Juliette is interested in having all the available information about Andrew, then Juliette may ask:

$$
\begin{array}{ll}
?\text{-} & KB\text{:}solve(X('Andrew)) \\
 & X = 'young\_man;\ X = 'student;\ \ldots
\end{array}
$$

Suppose now that Juliette wants to specialise the search in the database with respect to the properties she finds interesting in a man, for example, the following:

$$
\begin{array}{ll}
Juliette\text{:} & interesting(man, 'handsome) \\
 & interesting(man, 'rich)
\end{array}
$$

Then, she may decide to give him a date after having asked:

$$
\begin{array}{ll}
?\text{-} & KB\text{:}solve(X('Andrew)), Juliette\text{:}interesting(man, X) \\
 & X = 'handsome;\ X = 'rich
\end{array}
$$

A different kind of flexibility results from implementing, again by means of metaevaluation clauses, non-classical or non-deductive forms of reasoning, like for instance analogical reasoning, various forms of induction, etc.

6) Analogical reasoning/retrieval

6a) analogy as transfer of properties by similar hypotheses

$$
\begin{array}{ll}
KB\text{:} & solve('has\_property('x, Y)) \leftarrow similar(x, z), solve('has\_property('z, Y)) \\
 & has\_property(Juliette, 'beautiful) \\
 & similar(Cinderella, Juliette)
\end{array}
$$

$$
\begin{array}{ll}
?\text{-} & KB\text{:}has\_property(Cinderella, 'beautiful) \\
 & yes
\end{array}
$$

6b) analogy as transfer of properties by determination rules

$$
\begin{aligned}
\textit{KB:}\quad & solve(X(Z,L)) \leftarrow determines(Y,X), solve(Y(Z,N)),\\
& \qquad\qquad solve(Y(P,N)), solve(X(P,L))\\
& determines('nationality, 'native\_language)\\
& native\_language(Philippe, french)\\
& nationality(Jean, France)\\
& nationality(Philippe, France)
\end{aligned}
$$

$$
\begin{aligned}
\textit{?-}\quad & KB{:}native\_language(Jean, x)\\
& x = french
\end{aligned}
$$

7) Integrity constraints
Any person that is married is not unmarried.

$$
\begin{aligned}
\textit{KB:}\quad & married(Bob)\\
& solve\_not('unmarried(X)) \leftarrow solve('married(X))
\end{aligned}
$$

$$
\begin{aligned}
\textit{?-}\quad & KB{:}unmarried(Bob)\\
& no
\end{aligned}
$$

Note that this (metalevel) representation is such that the answer will satisfy the integrity constraint even if the data does not.

8) Cooperative answering
When we put a query to a database system, the system can provide additional information relevant to the query. [Cuppens, Demolombe, 1988] developed a method based on the use of topics. If the query is a sentence Q, and if Q is about the topic T, then T is identified as a topic of interest, and the system returns, in addition to the answer to Q, other sentences that are about that topic and that are consequences of the database. This idea can be formalized as follows:

$$
\begin{aligned}
\textit{KB:}\quad & topic(top1, 'sentenceA)\\
& topic(top1, 'sentenceB)\\
& topic(top2, 'sentenceC)\\
& sentenceB\\
& sentenceC\\
& solve(Q) \leftarrow topic(T,Q), topic(T,P), P \neq Q, solve(P), write(P)
\end{aligned}
$$

$$
\begin{aligned}
\textit{?-}\quad & KB{:}sentenceA\\
& \quad 'sentenceB\\
& yes
\end{aligned}
$$

Cuppens & Demolombe also discuss the possibility of defining a structure on sets of topics. We could for example formalise the fact that a topic *top1* is more specific than a topic *top2*, assume as $top1 < top2$. This allows us to represent hierarchies of topics, and it can be used to formalise statements as: if a topic T_1 is more specific than a topic T_2, and a sentence P is about T_1, then P is also about T_2.

$$\begin{array}{ll} KB: & topic(top1, 'sentence) \\ & top1 < top2 \\ & topic(T_2, P) \leftarrow topic(T_1, P), T_1 < T_2 \end{array}$$

9) Data protection

The notion of topic allows us to characterise the set of sentences an agent is permitted or prohibited to access in a database [Cuppens, Demolombe, 1988]. An agent may be permitted or prohibited to access all the data which are about some topic *top1* and *top2*.

$$\begin{array}{ll} KB: & topic(T_2, P) \leftarrow topic(T_1, P), T_1 < T_2 \\ & top1 < top2 \\ & topic(top1, 'sentenceA) \\ & topic(top2, 'sentenceB) \\ & pkt(Ralf, top1) \\ & pkt(John, top2) \end{array}$$

where $pkt(A, T)$ means that agent A is permitted to know all the sentences about topic T. Permissions to know or prohibitions to know may be represented by means of the following clauses.

$$\begin{array}{ll} KB: & pk(A, Q) \leftarrow topic(T, Q), pkt(A, T) \\ & pk(A, P) \leftarrow topic(T, P \text{ if } Q), pkt(A, T), pk(A, Q) \\ & pk(A, P \text{ and } Q) \leftarrow pk(A, P), pk(A, Q) \end{array}$$

where $pk(A, Q)$ means that agent A is permitted to know a sentence Q, and *if* and *and* are intended to name "$\leftarrow$" and ",", respectively.

$$\begin{array}{ll} ?\text{-} & KB{:}pk(Ralf, X) \\ & X = 'sentenceA \\ ?\text{-} & KB{:}pk(John, X) \\ & X = 'sentenceA; \ X = 'sentenceB \end{array}$$

1.4 FLEXIBLE QUERY ANSWERING IN DKB'S: THE ROLE OF REFLECTIVE AGENTS

In the context of knowledge bases, modularization capabilities are essential for providing a really flexible view of the knowledge base to the different kinds of users interacting with the system. In fact, each agent may employ different query-answering strategies. The agents then can either encapsulate or share (portions of) the DKB. We demonstrate through a number of examples how the interplay between metalogical reasoning and different modalities of communications allows us to obtain systems with a high degree of flexibility.

In the context of communication-based reasoning, the interaction among agents is based on communication acts. In particular, every agent can ask other agents questions in order to solve a given problem. Let *Bob*, *Anne* and *John* be theory symbols.

10) *Bob* tells *Anne* anything he can prove and lies to *John*.

$$\begin{array}{l} Bob : \mathit{tell}('Anne, '\mathit{pp}('x)) \leftarrow \mathit{pp}(x) \\ Bob : \mathit{tell}('John, '\mathit{not}\ '\mathit{pp}('x)) \leftarrow \mathit{pp}(x) \end{array}$$

11) *Bob* tells every agent anything he can prove within some resource limitations,

$$Bob : \mathit{tell}(A, X) \leftarrow \mathit{limited\_prove}(X)$$

where the predicate *limited_prove* incorporates the desired limitations.

12) *Bob* trusts every agent that is reliable but distrusts *John*.

$$\begin{array}{l} Bob : \mathit{solve}(X) \leftarrow \mathit{reliable}(A), \mathit{told}(A, X) \\ Bob : \mathit{solve}(X) \leftarrow \mathit{told}('John, '\mathit{not}\ X) \end{array}$$

The approach to inter-agent communication allows us to formalise a variety of interactions among agents depending on the application context. This is shown in the next examples.

13) The agent Bob tells the others whatever he can prove.

$$Bob : \mathit{tell}(A, X) \leftarrow \mathit{solve}(X)$$

14) *Bob* tells a group of agents whatever he can prove about a predicate *pp*, and another group of agents whatever he can prove about predicate *qq*.

$$\begin{array}{l} Bob : \mathit{tell}(A, '\mathit{pp}('x)) \leftarrow \mathit{group1}(A), \mathit{pp}(x) \\ Bob : \mathit{tell}(B, '\mathit{qq}('x)) \leftarrow \mathit{group2}(B), \mathit{qq}(x) \end{array}$$

15) *Bob* tells the others whatever he can prove about a predicate *pp* in a certain module *mod1*.

$$Bob : tell(A, {'}pp({'}x)) \leftarrow module(mod1, {'}pp({'}x))$$

16) *Bob* may ask other agents to answer a query according to which group the request is coming from.

$$Bob : tell(A, Q) \leftarrow group1(A), server({'}group1, S), told(S, Q)$$
$$Bob : tell(A, Q) \leftarrow group2(A), server({'}group2, S), told(S, Q)$$

17) *Bob* may also pass to the server the information about the original agent asking the query. This is achieved by means of an optional third argument to *tell*/*told*. In this way the server can decide locally what kind of answer to give according to where the original request comes from.

$$Bob : tell(A, Q) \leftarrow group1(A), server({'}group1, S), told(S, Q, A)$$

Users can be modeled as agents interacting with the DKB. Agents representing users can be divided into categories, or groups, corresponding, for instance, to: (i) their access privileges; (ii) the portion of the knowledge base they need to access; (iii) the view of the knowledge base they need to have. Then, also the format of the answer can be adapted to the needs or the skill of the particular user.

An agent implementing a user interface can route questions to other agents, associated with a specific portion of the DKB, and/or to different kind of answers to the same questions, according to the group the user belongs to.

The specification of the agents is in the usual style of Horn clause logic programming, since all these modalities of interactions are defined as metalogic axioms of the above form. They are applied (like Horn clauses) to *tell*/*told* subqueries, and, by means of reflection, result in the context-switching and information exchange among the involved agents.

1.5 EXAMPLE: AN INTELLIGENT TUTORING SYSTEM

Below we sketch the specification of an intelligent tutoring system able to provide students with exercises concerning specific topics they are interested in. The system has its own classification of the users, possibly on the basis of previously selected items.

The agent *interface* defines the user-interface of a system which receives requests for new exercises from the agents corresponding to the users. According to a table that associates students with topics, *interface* asks the agent managing that topic for the exercise, also specifying who is the user issuing the request.

The agent *maths* is an example of an agent managing a topic, i.e., mathematics, that selects an exercise from a suitable library according to the level of expertise of the user. The level (for instance beginner) and the name of the corresponding library (for example *libMB*, i.e. library of maths for beginners) are returned by suitable predicates. The library itself is represented by a predicate, which specifies the description of the exercises. This description is extracted by a predicate *select* which takes the name of the library as argument, and is able to actually access the library by means of a metaevaluation (*solve*) clause, that commutes from the name to the invocation of the predicate.

Finally, the agent corresponding to each user will be able, by means of its metalogic capabilities (as discussed in the previous section), to propose the exercise and check the result according to the needs of the particular user. For instance, the text of the exercise can be printed in the language spoken by that user, and can be possibly repeated by using synonyms whenever possible, in case the user does not understand.

$$\begin{array}{ll} \textit{interface:} & \textit{tell}(U,E) \leftarrow \textit{interested\_in}(U,T), \textit{told}(T,E,U) \\ & \textit{interested\_in}('Anne, 'maths) \\ & \textit{interested\_in}('George, 'maths) \\ & \textit{interested\_in}('Dan, 'computer\_science) \end{array}$$

$$\begin{array}{ll} \textit{maths:} & \textit{tell}(I,E,U) \leftarrow \textit{level}(U,L), \textit{library}(L,M), \textit{select}(E,M) \\ & \textit{level}('Anne, \textit{beginner}) \\ & \textit{level}('George, \textit{intermediate}) \\ & \textit{level}('Dan, \textit{expert}) \\ & \textit{library}(\textit{beginner}, 'libMB) \\ & \textit{library}(\textit{intermediate}, 'libMI) \\ & \textit{library}(\textit{expert}, 'libME) \\ & \textit{select}(E,M) \leftarrow \textit{solve}(M(E)) \\ & \textit{libMB}(ex(1, sub(equations), text(\lceil x-3=5 \rceil), res(\lceil x=8 \rceil))) \\ & \textit{libMB}(ex(2, sub(equations), text(\lceil x*x=16 \rceil), res(\lceil x=4 \rceil))) \\ & \ldots \end{array}$$

where $\lceil x-3=5 \rceil$ and $\lceil x*x=16 \rceil$ above are abbreviations for whatever names are used for these equations.

In the above example we use both agents and predicates. The difference in our view is that predicates have a fixed, non-interactive behaviour, according to the given arguments, while agents have a flexible, possibly interactive behaviour, depending on its argument which is the other agent asking with *told*. This agent may possibly be the user.

This is a main difference w.r.t. the object-oriented paradigm, where every entity must be an agent. In metalogic, we may have plain modules (predicates), like a library (say "inanimated entities"), as well as agents (say "animated entities"). As the example above shows, the two aspects coexist without problems. Also, we are not forced to introduce inheritance (although this is feasible in principle) whenever this is not necessary. In fact, inheritance is hardly necessary in query-answering systems where cooperation should be a more important feature.

1.6 RELATED WORK AND CONCLUDING REMARKS

The aim of this paper was to show that a language with simple metalogical and communication facilities, both based on a single notion of reflection with well-defined declarative and procedural semantics, is apt to cover a wide variety of flexible query-answering forms. These forms, which we have presented through sets of examples, touch upon a broad spectrum of topics that have been investigated within the areas of databases, knowledge-based systems, machine learning, agent-based programming, and others.

Though there are plenty of interrelationships among such topics, let us mention just a few of them. [Russel, 1989] has shown that determination rules (such as those used in example 6b) generalise the functional dependencies and extended multi-valued dependencies considered in database theory [Ullman, 1980], though for integrity checking rather than for reasoning. Integrity constraints themselvas (like those shown in example 7) have played a role both in database systems and in machine learning, expecially in the context of abductive logic programming (see e.g. [Kakas *& al*, 1993] for the latter). Compilation of rules for more efficient future query processing has been considered both in the context of machine learning (under the heading of Explanation-Based Learning, see [Ellman, 1989] for an overview) and of expert systems and databases (where the so-called semantic query optimization is viewed as a mechanism to separate the intensional from the extensional database components, see e.g. [Chakravarthy *& al*, 1986]). The EBL technique can be extended with metaknowledge, for answering queries that otherwise do not follow (like in [Bertarello *& al*, 1994]).

We hope to have convinced the reader that the degree of flexibility that can in principle be attained by means of a logical reflective language like the one shown in this paper is already high enough, so that the question is not how to improve it further. Rather, the main open problem is how to achieve it efficiently. Clearly, much further work is required to "rigidly" answer this question.

References

[Andreasen, Christiansen, 1996] Andreasen, T., Christiansen, H. An experimental prototype for flexible query-answering mechanisms. A metainterpretation approach. In: Christiansen, H., Larsen, H.L., Andreasen, T., editors, *Proc. of the 1996 workshop (FQAS96)*, Roskilde, Denmark, 1996. *Datalogiske skrifter* 62, Roskilde University, 1996.

[Barklund *& al*, 1994] Barklund, J., Costantini, S., Dell'Acqua, P., Lanzarone, G.A. SLD-resolution with reflection. In: M. Bruynooghe, editor, *Logic Programming – Proc. 1994 Intl. Symp.*, 554–568, Cambridge, Mass., 1994. MIT Press.

[Barklund *& al*, 1995] Barklund, J., Costantini, S., Dell'Acqua, P., Lanzarone, G.A. Semantical properties of encodings in logic programming. In: Lloyd, J., editor, *Logic Programming – Proc. 1995 Intl. Symp.*, 288–302, Cambridge, Mass., 1995. MIT Press.

[Bertarello *& al*, 1994] Bertarello, S., Costantini, S., Lanzarone, G.A. Extending explanation–based generalization with metalogic programming. In: Alpuente, M., Barbuti, R., Ramos I., editors, *Procs. GULP-PRODE Joint Conference on Declarative Programming*, vol. II, 16–31, 1994.

[Chakravarthy *& al*, 1986] Chakravarthy, U.S., Fishman, D.H., Minker, J. Semantic query optimization in expert systems and database systems. In: Kerschberg, L., editor, *Expert Database Systems*, 659–674, 1986. Benjamin/Cummings Publishing Company Inc.

[Costantini, 1990] Costantini, S. Semantics of a metalogic programming language. *Int. J. of Found. of Comp. Sci.*, 1, 1990.

[Costantini *& al*, 1992] Costantini, S., Dell'Acqua, P., Lanzarone, G.A. Reflective agents in metalogic programming. In: Pettorossi, A., editor, *Meta-Programming in Logic*, LNCS 649, 135–147, Berlin, 1992. Springer-Verlag.

[Costantini, Lanzarone, 1989] Costantini, S., Lanzarone, G.A. A metalogic programming language. In: Levi, G., and Martelli, M., editors, *Proc. 6th Intl. Conf. on Logic Programming*, 218–33, Cambridge, Mass., 1989. MIT Press.

[Costantini, Lanzarone, 1992] Costantini, S., Lanzarone, G.A. On procedural semantics of metalevel negation. In: Voronkov, A., editor, *Proc. 2nd Russian. Conf. on Logic Programming*, LNCS 592, Berlin, 1992. Springer-Verlag.

[Costantini, Lanzarone, 1994] Costantini, S., Lanzarone, G.A. Metalevel negation in non-monotonic reasoning. *Intl. J. of Methods of Logic in Computer Science*, 1, 1994.

[Costantini, Lanzarone, 1994] Costantini, S., Lanzarone, G.A. A metalogical programming approach: language, semantics and applications. *J. Exper. Theor. Artificial Intelligence*, 6, 1994.

[Cuppens, Demolombe, 1988] Cuppens, F., Demolombe, R. Cooperative answering: a methodology to provide intelligent access to databases. *Proc. 2nd Int. Conf. on Expert Database Systems*, 621–643, Virginia, 1988.

[Ellman, 1989] Ellman, T. Explanation-based learning: a survey of programs and perspectives. *ACM Computing Surveys*, vol.21, n.2, 163–221, 1989.

[Feferman, 1962] Feferman, S. Transfinite recursive progressions of axiomatic theories. *J. Symbolic Logic*, 27:259–316, 1962.

[Kakas *& al*, 1993] Kakas, A.C., Kowalski, R.A., Toni, F. Abductive logic programming. *Journal of Logic and Computation*, vol. 2, n. 6, 719–770, 1993.

[Kowalski, 1974] Kowalski, R.A. Predicate Logic as a Programming Language. In: Rosenfeld, J.L., editor, *Information Processing*, 569–574, Amsterdam, 1974. North-Holland.

[Lloyd, 1987] Lloyd, J.W. *Foundations of Logic Programming, Second Edition.* Springer-Verlag, Berlin, 1987.

[Loo *& al*, 1994] Loo, G.S., Dillon, T., Zeleznikow, J., Lee, K.H. Two approaches for answering inadequate queries - Empirical projectIdb-Kroom. *Proc. of the 1994 workshop (FQAS94)*, Roskilde, Denmark, Nov. 14–16, 1994. *Datalogiske skrifter* 58, Roskilde University, 1995.

[Russel, 1989] Russel, S.J. The use of knowledge in analogy and induction. Pitman – Morgan Kaufmann, 1989.

[Ullman, 1980] Ullman ,J.D. Principles of data base systems. Rockville, Computer Science Press, 1980.