Semantic Web Reasoning for Ontology-based Integration of Resources

Liviu Badea [1], Doina Tilivea [1], Anca Hotaran [1]

[1]AI Lab, National Institute for Research and Development in Informatics
8-10 Averescu Blvd., Bucharest, Romania
{badea,doina,ahotaran}@ici.ro

Abstract. The Semantic Web should enhance the current World Wide Web with reasoning capabilities for enabling automated processing of possibly distributed information. In this paper we describe an architecture for Semantic Web reasoning and query answering in a very general setting involving several heterogeneous information sources, as well as domain ontologies needed for offering a uniform and source-independent view on the data. Since querying a Web source is very costly in terms of response time, we focus mainly on the query planner of such a system, as it may allow avoiding the access to query-irrelevant sources or combinations of sources based on knowledge about the domain and the sources.

Taking advantage of the huge amount of knowledge implicit and distributed on the Web is a significant challenge. The main obstacle is due to the fact that most Web pages were designed for human-centred browsing rather than being machine-processable. In addition to static HTML pages the Web currently offers online access to a large number information resources, such as databases with a Web interface. But real-life applications frequently require combining the information from several such resources, which may not have been developed with this interoperability requirement in mind. Thus, a large amount of knowledge is implicit, heterogeneously distributed among various resources and thus hard to process automatically.

The recent developments towards a "Semantic Web" should help address these problems. Being able to explicitly represent domain-specific knowledge in the form of ontologies, should allow reasoning about such machine-processable Web pages.

The emergence of standards for data markup and interchange such as XML and for representing information about resources and their semantics (such as RDF and RDF Schema) can be seen as a first step in the transition towards a Semantic Web. However, the vast majority of Web pages still conform to the HTML standard, which only controls their visual aspects rather than their informational content. Extracting the informational content from such pages which essentially contain free text is a difficult practical problem. The Resource Description Framework (RDF) has been designed to complement such human-oriented text with machine-processable annotations. A large number of prototype systems able to read and reason about such annotations have been developed (TRIPLE [7], Metalog [20], SiLRI [8], Ontobroker [9]). However, currently only a very small minority of Web pages have RDF annotations. Moreover, existing annotations tend to refer to basic features such as document author, creation date, etc., but do not duplicate the information content of the page.

On the other hand, a large number of information sources have a Web interface and could be the building blocks of complex applications, were it not for the unavoidable semantic mismatch between such resources developed by different people. Such Web interfaces produce pages with a partially stable structure, so that their content can be automatically extracted using wrappers, thus replacing human annotation (which is a significant bottleneck in practice).

Dealing with information sources rather than a fixed set of Web pages may pose additional problems. For example, systems like TRIPLE read *all* the relevant (RDF) annotations *before* reasoning about them. In the case of large data sources however, it is obviously impossible to retrieve the *entire* content of such sources before starting reasoning. Also, if additional knowledge is available about the sources, some source accesses may be avoided altogether. Therefore, dealing with information sources requires a certain form of *query planning*, i.e. the ability of constructing and reasoning about alternative sequences of source accesses (plans) before actually querying these sources. Also, *streaming the query responses* may allow starting processing before the entire content of the information source is retrieved.

In this paper we present an approach to such more complex Semantic Web scenarios, involving the integration of heterogeneous resources using rules and ontologies.

The most significant problem faced when trying to combine several resources is related to their heterogeneity. This heterogeneity can be either *structural* (different schemas), or *semantic* (the same entity can be represented using different terms from different vocabularies). Integrating such resources can be achieved by mapping them (both their schemas and their content) to a *common* "knowledge" level, at which their interoperation is straight-forward. This common level involves not just a common (domain-specific) vocabulary, but also formal (machine processable) descriptions of the terms of this vocabulary, as well as the relationships between them, which form a so-called *ontology*. A *mediator architecture* [22] can be used for query answering.

A researcher specialized in Knowledge Representation and Reasoning (KRR) might be very disappointed by current Web technology, which:
- involves to a large extent HTML pages in mostly free text (not machine processable)
- the knowledge is not well structured
- there is virtually no support for reasoning.

Thus, since current state of the art in Natural Language Processing does not allow extracting the deep semantics of free text, the temptation may be very high to change everything. However, as it isn't easy to impose a radically new (even if better) Web standard, we adopt an *evolutionary* rather than revolutionary approach and conform as much as possible to current and emerging Web standards.

In the following we concentrate mainly on the query planning component of a mediator-based Semantic Web architecture – other important issues such as wrappers and especially ontologies deserve a more detailed discussion, but which is outside the scope of this paper.

1 The architecture: public and private levels

The distinctive feature of the Semantic Web is reasoning. However, the various W3C standards related to the (Semantic) Web are not easy to use by a reasoner, especially due to their heterogeneity (XML, RDF, RuleML, etc.). A *uniform* internal level would be much more appropriate for supporting inference and reasoning. The architecture of our system therefore separates a so-called *"public" level* from the *internal level.* The public level refers to the data, knowledge and models exchanged on the Web and between applications and must conform to the current and emerging Web standards such as XML, RDF(S), RuleML, etc.

1.1 The internal representation

We use *F-logic* [12] for describing the content of information sources as well as the domain ontology for several important reasons. First, a logic-based language is not only declarative, but also offers support for reasoning. However, a Prolog-like syntax using predicates with a fixed number of arguments and a position-oriented argument access is not well suited for dealing with the semi-structured data available on the Web. On the other hand, F-logic combines the logical features of Prolog with the frame-oriented features of object-oriented languages, while offering a more powerful query language (allowing e.g. aggregation and meta-level reasoning about the schema). Last but not least, F-logic is widely used in the Semantic Web community [7,18,8,9].

In the following (and throughout the whole paper) we use the term "*predicate*" to denote an F-logic molecule, such as *X:c*[*a1–>Y1*, …].

While the data content of an information source can be represented by a number of so-called *source predicates s*, the user might prefer to have a *uniform* interface to *all* the sources instead of directly querying the sources *s*. In fact, she may not even *want* to be aware of the structure of the information sources. Therefore, the mediator uses a uniform knowledge representation language in terms of so-called *"model predicates"*, which represent the user's perspective on the given domain and refer to concepts and terms of an associated domain ontology.

There are two main viewpoints on representing sources and their interactions: *"Global as View"* (GAV) and *"Local as View"* (LAV) [16]. The query planner presented in this paper can deal with both GAV and LAV approaches. However, we encourage the use of LAV, since it spares the knowledge engineer the effort of explicitly determining and representing the source interactions.

1.2 Source predicates, model predicates and description rules

The content of the various Web information sources is represented by so-called *source predicates.* For a uniform and integrated access to the various sources these are described (at the mediator level) in terms of so-called *model predicate*s.

More precisely, we distinguish between *content* predicates and *constraint* predicates.

Content predicates (denoted in the following with *p*, *q*) are predicates which directly or indirectly represent the *content* of information sources. They can be either *source* predicates or *model predicates*.

Source predicates (denoted with *s*) directly represent the content of (part of) an information source (for example, the content of a Web page or the answer returned by a Web service).

Model predicates (denoted with *b*) are used to describe the "global model", including the domain ontology and the information content of the sources in terms of this ontology.

As opposed to content predicates, *constraint predicates* (denoted by *c*) are used to express specific constraints on the content predicate descriptions.

For example, a source *s* containing information about underpaid employees (with a salary below 1000), would be described as:

E:s[*name–>N, salary–>S*] —> *E:employee*[*name–>N, salary–>S*], *S<1000.*

Constraint predicates can be either *internal* (treatable internally by the query engine of the source), or *external* (constraints that can only be verified at the mediator level, for example by the built-in constraint solvers of the host CLP environment). Constraints treatable *internally* by the sources can be *verified* at the source level (by the query engines of the sources), but they are also *propagated* at the mediator (CLP) level. Constraints treatable only *externally* need to be both verified and propagated at the mediator level.

A *complete* description of the source predicates in terms of model predicates is neither possible nor useful, since in general there are too many details of the functioning of the application. Thus, instead of *complete* (iff) descriptions, we shall specify only approximate (necessary) definitions of the source predicates in terms of model predicates (thus, only the relevant features of the sources will be encoded).

In the following, we use a *uniform* notation for the domain and source ***description rules***:

$$Antec, Constr_a \rightarrow Conseq, Constr_c \qquad \text{(dr)}$$

where *Antec* and *Conseq* are conjunctions of content predicates, while $Constr_a$ and $Constr_c$ are conjunctions of constraints. Variables occurring in the consequent but not in the antecedent are implicitly existentially quantified. As in Prolog and F-logic, all other variables are implicitly universally quantified.

Description rules are necessary definitions of (combinations of) source or model predicates in terms of model predicates and constraints. (*Source descriptions* are special cases of description rules where the antecedent contains only source predicates. *Integrity constraints* are description rules with only constraints – typically *'fail'* – in the consequent.)

1.3 Querying the sources: wrappers

While the traditional Web was designed mainly for browsing, the Semantic Web should enable automated processing of Web sources. A Semantic Web system should therefore be able to deal not just with *static* Web pages (in various formats such as HTML, XML), but also with *dynamically generated* pages as well as Web services.

While the semantics of static Web pages is encoded using static annotations (e.g. in RDF) the information content of dynamically generated Web pages needs to be extracted by automatic *wrappers.* These have to be quite flexible, i.e. able to deal with XML files, or even possible non-well-formed HTML. Due to its wide spread use worldwide, we currently employ *XQuery* for implementing our wrappers (we also use *tidy* [1] to preprocess non-well-formed HTML sources).

2 Ontologies and mapping rules

A key component in a SW application is a *domain ontology*, used by a certain community to enable sharing and exchange of knowledge in a particular domain. Due to the heterogeneous nature of the sources, mapping rules from the various sources to the common ontology are needed for obtaining a unified view of these sources (see Section 4 below for examples).

OWL (http://www.w3.org/2004/OWL/) has recently emerged as a de-facto standard for Web ontologies. Since OWL is a description logic-based extension to RDF(S), it provides useful inference services, mainly for checking the consistency of a schema as well as for subsumption testing. However, description logics (DL) with tractable inference services tend to have limited expressiveness, so that they are closer to a dynamic type checker rather than to a more expressive reasoner. A large part of the semantics encoded in such an OWL ontology thus resides “in the names” rather than in the formulas themselves, thereby making the usefulness of the reasoning services questionable w.r.t. the limited expressiveness and the computational overhead (dealing with instances in current DL implementations is especially inefficient). Also, most real-life ontologies require the use of rules and domain-specific constraints for supplementing the limited expressiveness of description logics. However, combining DLs with rules is highly non-trivial, both theoretically and from the point of view of tractability [18]. A pragmatic approach would avoid the computational complexities of a *complete* DL reasoner by implementing a fast, albeit incomplete set of DL propagation rules.

We currently employ Protégé[2], a widely used Ontology development tool. Ontologies are exported in RDF(S) format and automatically converted to the internal (F-logic) format. As discussed above, additional rules may be needed to extend the expressiveness of RDF(S).

[1] http://tidy.sourceforge.net/

[2] http://protege.stanford.edu

3 Reasoning and Query Planning

Since querying a Web source is extremely costly in terms of response time, every possible optimization that might avoid accessing irrelevant sources should be attempted. The knowledge available about a particular source or combination of sources may imply the inconsistency of some potential query plan even *before* accessing the sources. The purpose of the query planner is to avoid executing such inconsistent plans and possibly to use additional cost measures for ranking plans.

The majority of Semantic Web implementations using F-logic [7,18,8,9] retrieve the content of *all* the sources before query answering. However, in applications in which there are alternative plans for a given query, some of these plans may be inconsistent and - for efficiency - the associated sources should not be accessed. This cannot be achieved simply by the normal F-logic query answering mechanism and requires some form of meta-interpretation.

Query planning amounts to unfolding the query in terms of sources and propagating the relevant constraints to eliminate the inconsistent plans. Since this type of reasoning is performed *before* accessing the sources, it can be viewed as an *abductive procedure* [14] in which the source predicates play the role of abducibles.

Due to their flexibility and declarative nature, Constraint Handling Rules (CHRs) [10] represent an ideal framework for implementing the reasoning mechanism of the query planner.

Constraint Handling Rules (see [10] for more details) represent a flexible approach to developing user-defined constraint solvers in a declarative language. As opposed to typical constraint solvers, which are black boxes, CHRs represent a 'no-box' approach to CLP.

CHRs can be either *simplification* or *propagation* rules.

A *simplification* rule *Head* <=> *Guard* | *Body* replaces the head constraints by the body provided the guard is true (the *Head* may contain multiple CHR constraint atoms).

Propagation rules Head ==> *Guard* | *Body* add the body constraints to the constraint store without deleting the head constraints (whenever the guard is true). A third, hybrid type of rules, *simpagation rules* $Head_1 \setminus Head_2$ <=> *Guard* | *Body* replace $Head_2$ by *Body* (while preserving $Head_1$) if *Guard* is true. (Guards are optional in all types of rules.)

CHRs allow us to combine in an elegant manner the backward reasoning necessary for implementing query unfolding with the forward propagation of abducibles and constraints.

A description rule of the form (dr) will be encoded in CHR using

- *goal regression rules*: for reducing queries given in terms of model predicates to queries in terms of source predicates,

$$\boldsymbol{G}\ Conseq \text{ :- } \boldsymbol{G}\ Antec,\ Constr_a \qquad \text{(b)}$$

- *propagation rules*: for completing (intermediate) descriptions in order to allow the discovery of potential inconsistencies.

$$\boldsymbol{H}\ Antec\ \texttt{==>}\ Constr_a\ \mid\ \boldsymbol{H}\ Conseq, Constr_c \qquad \text{(f)}$$

In our CHR embedding of the *abductive procedure* we use two types of constraints, $\boldsymbol{G}p$ and $\boldsymbol{H}p$, for each predicate p. While $\boldsymbol{H}p$ represents facts explicitly propagated (abduced), $\boldsymbol{G}p$ refers to the *current closure* of the predicate p (i.e. the explicit definition of p *together* with the explicitly abduced literals $\boldsymbol{H}p$).

While propagating $\boldsymbol{H}p$ amounts to simply assuming p to hold (abduction), propagating $\boldsymbol{G}p$ amounts to trying to prove p either by using its definition $def(p)$, or by reusing an already abduced fact $\boldsymbol{H}p$. In fact, our description rules can be viewed as a generalization of abductive logic programs with integrity constraints interpreted w.r.t. the 'propertyhood view'[3].

A set of subgoals in terms of source predicates (induced by backward chaining from the user query using the goal regression rules) may not necessarily be consistent. Applying forward propagation rules ensures the completion ("saturation") of the (partial) query plan and enables detecting potential conflicts *before* actually accessing the sources.

$Constr_a$ and $Constr_c$ in (f) represent constraint predicate calls, for which the associated constraints solvers are assumed to be available. The occurrence of $Constr_a$ in the guard ensures that the consequent is propagated only if $Constr_a$ hold.

Source and domain models are described using rules of the form (dr), which are then automatically translated by the system into CHR goal regression and propagation rules (b) and (f). The following *additional problem-independent rules* (gh) and (re) are used for obtaining the complete CHR encoding of a model.

- a CHR simpagation rule[4] for matching a goal $\boldsymbol{G}p$ with any existing abduced facts $\boldsymbol{H}p$:

  $$\boldsymbol{H}p(X_1) \setminus \boldsymbol{G}p(X_2)\ \texttt{<=>}\ X_1{=}X_2\ \mathbf{;}\ X_1{\neq}X_2, \boldsymbol{G}p(X_2). \qquad \text{(re)}$$

  This rule should have a higher priority than the unfolding rule in order to avoid reachieving an already achieved goal. Note that, for completeness, we are leaving open the possibility of achieving $\boldsymbol{G}p(X_2)$ using its definition or reusing other abduced facts.
- a rule taking care of the consistency between goals and facts that simply hold (since goals will be achieved eventually, they also have to hold):

  $$\boldsymbol{G}p\ \texttt{==>}\ \boldsymbol{H}p \qquad \text{(gh)}$$

We have already mentioned the fact that we have to distinguish between goals involving a given predicate p (which will be treated by a mechanism similar to the normal Prolog backward chaining mechanism) and the instances $\boldsymbol{H}p$ of p, which trigger forward propagation rules. Operationally speaking, while goals $\boldsymbol{G}p$ are "consumed" during goal regression, the fact that p holds should persist even after the goal has been

---

[3] The detailed presentation of the semantics is outside the scope of this paper and will be pursued elsewhere. Briefly, the goal regression rules make up the program P, the sources are regarded as (temporary) abducibles A, while the forward propagation rules play the role of the ALP integrity constraints I. Our notion of integrity constraints is however more general than that used in ALP.

[4] The rule is more complicated in practice, due to implementation details.

achieved, to enable the activation of the forward propagation rules of *p*. Note that rule (gh) should have a higher priority than (b) to allow goals *p* to propagate ***H****p* before applying the goal regression rules for *p*.

Certain forward propagation rules (f) may propagate facts that may never be involved (directly or indirectly) in conflicts. These facts may be very useful to the end user in certain applications, especially whenever the user would like to see all the known facts about the instances returned by the query. However, in other applications, we may wish to refrain from propagating such facts that may never lead to an inconsistency (detected by an integrity constraint). In this case, we perform a static dependency analysis of the rules, and generate forward propagation rules (f) only for predicates that may propagate an inconsistency.

4 An example

In this paper, we consider a hardware configuration problem as a typical example. A component integrator selling customized computer configurations may use components from several component providers, while trying to satisfy a set of compatibility and user constraints. This problem has many typical features of the Semantic Web:

- it involves distributed and dynamically changing information sources (the component catalogs of the different providers available via Web interfaces)
- the information is semantically heterogeneous, requiring a domain ontology for a uniform access
- the domain involves complex compatibility constraints between components, requiring constraint propagation during query planning
- there are several alternative sources for some given component, which makes query planning necessary.

Here we consider just a fragment of this hardware configuration domain, in order to emphasize the main features of our architecture. Assume we have two main component providers, or vendors, *flamingo* and *oktal*. (Since their catalogs are accessible via the Web in HTML format, a wrapper is used for extracting the relevant information from these Web pages.) A user query may ask for a system with an upper bound on the total price. Here we just concentrate on a "simple system" containing just a motherboard and a processor. Of course, the two components must satisfy a compatibility constraint (e.g. one should not attempt to use an AMD processor on an Intel-Pentium compatible motherboard).

Since the sources are heterogeneous, we first use so-called "mapping rules" to describe their content in terms of the ontology. For example, motherboards sold by *flamingo* are mapped onto the ontology concept *motherboard* (note that here we have a trivial mapping of all slots, with an additional *m_vendor* slot recording the name of the vendor): [5]

[5] In the following, we use the more concise F-logic syntax for rules. As already discussed above, the actual syntax will involve an XML encoding of a rule markup language, such as *RuleML* (http://www.ruleml.org) or *Xcerpt* [4].

Mb:fl_motherboard[*A–>X*] –—> *Mb:motherboard*[*A–>X,m_vendor–>flamingo*] (r1)

Any additional knowledge about the content of the sources may be very useful during query planning for discarding inconsistent plans even before accessing the sources. For example, we may know a lower bound for the price of a *flamingo* motherboard. We may also know that *flamingo* distributes only *intel* and *msi* boards. This is specified using the following source description rules:

Mb:fl_motherboard[*m_price–>P*] –—> *P* ≥ *70.* (ic1)
Mb:fl_motherboard[*brand–>Br*] –—> *member*(*Br*, [*intel, msi*]). (ic2)

We have similar rules for *oktal* motherboards, as well as for processors (available also from both *flamingo* and *oktal*):

Mb:okt_motherboard[*A–>X*] –—> *Mb:motherboard*[*A–>X, m_vendor–>oktal*]. (r2)
Mb:okt_motherboard[*m_price–>P*] –—> *P* ≥ *70.* (ic3)
Mb:okt_motherboard[*brand–>Br*] –—> *member*(*Br*, [*gigabyte, msi*]). (ic4)

Pr:fl_processor[*A–>X*] –—> *Pr:processor* [*A–>X, p_vendor–>flamingo*]. (r3)
Pr:fl_processor[*p_price–>P*] –—> *P* ≥ *150.* (ic5)

Pr:okt_processor[*A–>X*] –—> *Pr:processor*[*A–>X, p_vendor–>oktal*]. (r4)
Pr:okt_processor[*p_price–>P*] –—> *P* ≥ *150.* (ic6)

Note that the source descriptions (ic1), (ic2), (ic3), (ic4), (ic5) and (ic6) express knowledge about the content of the sources. (These descriptions could either be provided by the knowledge engineer or could be automatically retrieved from previous source accesses.)

However, answering queries frequently requires knowledge about the sources that cannot be inferred from their content alone. For example, *oktal* offers discounts for purchases over a given threshold, while *flamingo* does not:

oktal:vendor[*name–>oktal, discount–> 0.1, discount_threshold–>200*]. (r5)
flamingo:vendor[*name–>flamingo, discount–> 0, discount_threshold–>0*]. (r6)

The domain ontology encodes our knowledge about this particular domain, such as the concept hierarchy, descriptions of slots/attributes (type, cardinality constraints, etc.), and any additional description rules involving these concepts. For example, *motherboard*s and *processor*s are sub-concepts of *component*:

motherboard :: *component.*
processor :: *component.*

For an example of an ontology description rule, we may describe a simple *system* as a set of matching components, such as a *motherboard* and a *processor*. (Note the compatibility constraint between *motherboard* and *processor*: '*Pr.id = Mb.supported_CPU*'.)

Mb:motherboard, Pr:processor, Pr.id = Mb.supported_CPU –—> (r7)
X:system[*motherboard–>Mb, processor–>Pr*].

To show the functioning of the query planner, we consider the following user query which asks for a system with an upper bound of *210* on the (possibly discounted) price:

?- *S:system*[*motherboard–>Mb*[*brand = gigabyte*], *processor–>Pr*], (q1)
compute_discounted_price(*S, S_price*), *S_price ≤ 210*.

For simplicity, the computation of the discounted price is performed by *compute_discounted_price*(*S, S_price*).

The query is answered by first invoking the query planner, and subsequently executing the resulting plans. As discussed previously, query planning *unfolds* the query in terms of source calls while *propagating constraints* with the purpose of discarding the inconsistent plans before the actual source accesses. Note that constraint propagation is interleaved with query unfolding, so that inconsistencies are detected as early as possible.

In our example, query planning starts by unfolding the query (q1) to the following subgoals and constraint (as discussed previously, ***G*** stands for '*goal*', while ***H*** stands for '*holds*'):

G*S:system*[*motherboard–>Mb*[*brand = gigabyte*], *processor–>Pr*] (g1)
G*compute_discounted_price*(*S, P*) (g2)
S_price ≤ 210 (c1)

Then (g1) will be unfolded with (r7-B) [6] to:

G*Mb:motherboard*[*brand = gigabyte*] (g3)
G*Pr:processor* (g4)

Since motherboards can be acquired either from *flamingo* or from *oktal*, (g3) will be unfolded first with (r1-B) to the following (we first try to acquire the motherboard from *flamingo*):

G*Mb:fl_motherboard*[*brand–>gigabyte, m_price–>MP*, …] (g5)

This will propagate (with rule (gh)) ***H****Mb:fl_motherboard*[*brand–>gigabyte, m_price–>MP*,…] which in turn will activate (r1-F) and propagate ***H****Mb:motherboard*[*m_vendor–>flamingo, brand–>gigabyte*, …]. (h1)

But now, (h1) will trigger the (ic2) integrity constraint, which fails because the required brand for the motherboard (*gigabyte*) is not included in the list of distributed brands of the vendor *flamingo*.

Alternatively, we could obtain a motherboard from *oktal*, i.e. unfold ***G****Mb:motherboard* (g3) with (r2-B) to:

G*Mb:okt_motherboard*[*brand–>gigabyte, m_price–>MP*, …] (g5)

[6] (r*i*-B) is the goal reduction rule (*backward* version) of the rule (r*i*), while (r*i*-F) is the associated *forward* propagation rule.

H*Mb*:*okt_motherboard*[*brand–>gigabyte*, *m_price–>MP*, …] (h2)

is propagated with rule (gh). In this case, (r2-F) will propagate

H*Mb:motherboard*[*m_vendor–>oktal*, *brand–> gigabyte*, …]. (h3)

(h2) will trigger (ic2), which will be consistent because the brands known to be distributed by *oktal* are *gigabyte* and *msi*. (h3) will also trigger (ic3) and propagate the price constraint $Mb.m\_price \geq 70$ (c2).

Having selected a consistent source for the motherboard, we now consider selecting a processor. The goal (g2) ***G****Pr:processor* will be first unfolded using (r3-B) to:

G*Pr:fl_processor*[*p_price–>PP*,…] (g6)

and will propagate with (gh) and (r3-F):

H*Pr:fl_processor*[*p_price–>PP*] (h4)

which will trigger (ic5) and propagate the price constraint $Pr.p\_price \geq 150$. (c3)

Then the next goal, ***G****compute_discounted_price*(*S, S_Price*) (g2), is unfolded and partially evaluated. For the simplicity of the presentation, we do not present here all the intermediate steps. Briefly, although at query planning time the prices of the components are not yet known, we can nevertheless propagate the constraints (c2) and (c3) and thus obtain a lower bound on *S_price*.

More precisely, since we have different vendors for the two components (*Pr.p_vendor = flamingo*, *Mb.m_vendor = oktal*), any potential discounts will be applied separately to each component. But *flamingo* doesn't offer discounts and the discount threshold for *oktal* is too high (*200*) to be applicable. Therefore, the system price will be: $S\_price = Pr.p\_price + Mb.m\_price \geq 220$ (due to (c3) and (c2)), which is inconsistent with the (c1) constraint $S\_price \leq 210$, so the combination of an *oktal* motherboard and a *flamingo* processor will be rejected.

Alternatively, we could acquire the processor from *oktal*, i.e. reduce ***G****Pr:processor* (g1) with (r4-B), (r4-F) and (gh) to:

H*Pr:okt_processor*[*p_price–>PP*, ..] (h5)

As before, (h5) will trigger (ic6) and propagate the price constraint $Pr.p\_price \geq 150$. (c4)

But now the unfolding of *compute_discounted_price*(*S*, *S_price*) (g2) will be different, as both components are acquired from the same vendor (*Pr.p_vendor = oktal*, *Mb.m_vendor=oktal*), so that the discount is applicable to both components. In fact, since the undiscounted price $S\_price1 = Pr.p\_price + Mb.m\_price$ is above the discount threshold for *oktal* $S\_price1 \geq 220$ (due to (c4) and (c2), *oktal.discount_threshold = 200*), the system price will be: $S\_price = S\_price1 \cdot (1 - oktal.discount) = 0.9 \cdot S\_price$. Thus, the lower bound on *S_price* will be *198*, which is consistent with the (c1) constraint. Query planning has therefore retained only the following combination of sources: *okt_processor* and *okt_motherboard*, while discarding the other three possible combinations at planning time, so that only *okt_processor* and *okt_motherboard* are actually queried.

5 Source Capabilities

Information sources are viewed as *collections of source predicates* that can be accessed via a specialized query interface. The query planner reduces a query formulated in terms of model predicates to a query in terms of source predicates and constraints. However, since such a "global" query can involve source predicates from *several* information sources, it will have to be to *split* into sub-queries that can be treated by the separate information sources. Since each information source may have its own Web interface, we need to explicitly represent the *capabilities* of these interfaces. As opposed to traditional database query languages, such Web sources provide only limited query capabilities. For example, a specific Web interface may allow only certain types of selections and may also require certain parameters to be inputs (i.e. known at query time).

More precisely, the capabilities of a given information source are described using the following items:

- the *name* of the source
- the *source predicates* that can be accessed from this source, with (optional) input argument annotations (these can be Web pages, Web interfaces to databases, or even Web services)
- the *constraints* that can be treated (internally) in selections (as filters).

Parameters annotated with '+' denote inputs, assuming that

- input parameters have to be instantiated at query time
- the other parameters are completely uninstantiated at query time

The input-output specifications determine the dataflow in the query. Such specifications are especially important in the case of Web interfaces to databases, as well as for describing Web services. The plans produced by the query planner have to be refined in order to conform to these source capabilities.

We define a *precedence* relation over the variable occurrences in source predicates.

A variable occurrence *X directly precedes* another variable occurrence *Y* iff *X* occurs in a predicate that provides an input to a + variable of another predicate containing *Y* (e.g. $p(X,\boldsymbol{W})$, $q(\boldsymbol{+W},Y)$). The '*precedes*' relation is the transitive closure of the '*directly precedes'* relation.

We also say that a variable *X precedes* a predicate (literal) *p* iff *X* precedes some variable occurrence *Y* in *p*. We have similar notions of predicates preceding variables or other predicates.

Let $p(\overline{X})$ be a literal in a query plan. Since the + variables of *p* have to be instantiated before calling *p*, we have to make sure that all predicates preceding *p* have been called already.

A query plan is *consistent* w.r.t. the source capabilities iff the precedence relation on its literals is acyclic.

Since on the Web it will be virtually impossible to retrieve all records of source predicates with a very large number of such records, we additionally categorize sources as being either "*large*" or "*normal*".

Starting from a "logical" plan, we first try to assign to all "large" predicates binding patterns[7] with as many input(+) arguments as possible, while keeping the query plan consistent (i.e. acyclic). Once having assigned binding patterns to all predicates of the plan, we execute it (without considering later on alternative binding pattern assignments, since the different binding patterns only control the format of the query and not its answer set).

To further reduce the number of tuples retrieved from the sources (this being the most important efficiency bottleneck), we may take advantage of the sources that allow adding filters to the query (e.g. ask about motherboards *with prices bellow 200*). These *filters* can be constructed by propagating the user constraints from the query with other constraints from the source descriptions and the domain ontology. Of course, we can use as filters propagated constraints that refer exclusively to variables from the given predicate $p(\overline{X})$ and that match the source's capabilities.

However, there is an additional subtlety related to the necessity of making as few source calls as possible.

For example, in the case of the following query:

?- *p*(*X*, *Y*), *s*(*Z*), *q*(+*Y*, *V*), *Y*+*Z*+*V* < *200*, *Y* > *0*, *Z* > *0*, *V* > *0*.

we have to execute *p* before *q* in order to instantiate *Y* for *q*, but *s* can be called at any time. If *s* is called earlier than *q* (which is a natural thing to do if parallel plan execution is possible), then *Z* will be instantiated at the time of *q*'s call.

Naively, one would be tempted to use all the current variable instantiations (say *Y=60*, *Z=100*) to obtain a reduced constraint *V*<40 to be used as a filter in the call *q*(*Y*=60, *V*), *V* < 40. However, every instantiation of *Z* will produce a different call to *q* and thus we would have to call *q* a large number of times. Even worse, the sets of tuples returned, e.g. by *q*(*Y=60*, *V*), *V* < *40*; *q*(*Y=60*), *V<100*; etc. will (partially) overlap.

A better solution would be to propagate only the instantiations of the variables that (necessarily) precede *q*, i.e. *Y=60*, but not *Z*. In this case, the propagated constraint *V<140* may be weaker, but it will be *the same for all bindings of Z*. Thus, a single query to *q*: *q*(+*Y=60*, *V*), *V<140* will be enough, since we can reuse the tuples returned by it for all bindings of *Z*.

More precisely, let C be the set of constraints propagated from the user constraints in the query. When executing the source predicate call $p(\overline{X})$, we attach to it as filter the set of constraints $C(\overline{X}, \overline{X}'=\overline{x}')|_{SC}$ obtained by propagating from C the instantiations $\overline{X}'=\overline{x}'$ of the variables $\overline{X}'$ that precede *p* and retaining only those constraints that satisfy the source capabilities SC.

---

[7] A given source predicate can have several input-output *binding patterns*. For example *pubmed*(+*PMID*, *Title*, *Author*), *pubmed*(*PMID*, +*Title*, *Author*), *pubmed*(*PMID*, +*Title*, +*Author*), etc.

6 Conclusions and Future Work

An exhaustive comparison with other information integration systems is impossible, due to the very large number of such systems as well as to the lack of space. Briefly, while database oriented approaches to integration (such as *multi-databases* and *federated databases*) use *fixed* global schemas and are appropriate only if the information sources and users do not change frequently, we deal with *dynamically evolving* schemas (especially if the LAV modeling approach is employed). On the other hand, more *procedural* intelligent information integration approaches, like TSIMMIS [11] and even some declarative systems like MedLan [1] or HERMES [21], use explicit query reformulation rules[8], but *without the equivalent of our forward propagation rules* (which allow an early discovery and pruning of inconsistent plans before query execution). Our approach is closer to the more declarative systems like SIMS [2], Information Manifold [17] and Infomaster [6].

COIN [5] also uses a CLP framework (Eclipse) for abductive reasoning and CHRs for implementing integrity constraints. However, integrity constraints can be imposed in COIN *only on source predicates*. Thus, COIN domain knowledge reduces to Prolog reduction rules, which are used *only backwards* (during goal regression). The lack of forward propagation rules involving base predicates (and not just sources) makes the discovery of potential interactions between base predicates (and thus the full use of domain knowledge) impossible. Other related mediator-based Web integration systems are EMERAC [13] and Ariadne [15].

This paper extends our research related to the SILK intelligent information environment [3] to deal with the specificities of the Semantic Web. (The semi-structured nature of the data on the Web lead us to the use of an F-logic based internal reasoning level, while the limited source capabilities of Web resources and the high access costs required a different query planning strategy). A prototype implementation using plain Prolog (rather than F-logic) and CHR already exists. However, as shown in this paper, F-logic is a much more appropriate internal language for dealing with semi-structured data. We are currently working towards extending our system to use F-logic as internal reasoning language. This is non-trivial as there are no Prolog environments allowing a combination F-logic and CHR.[9] In this sense, our work is related to Xcerpt [4], an elegant declarative, rule-based query and transformation language for XML, for which a query planner is also currently under development.

Acknowledgements

We are grateful to François Bry and Sebastian Schaffert for insightful discussions. The present work has been partially supported by the REWERSE Network of Excellence of the European Community (http://www.rewerse.net).

[8] Such query templates correspond to our goal regression rules.

[9] The upcoming XSB Prolog release should contain both Flora2 and CHR, but their combination may not be straightforward (if at all possible).

References

1. Aquilino D., Asirelli P., Renso C., Turini F. MedLan: a Logic-based Mediator Language, IEI Technical Report B4-16, November 1997.
2. Arens Y., Knoblock C.A., Chun-Nan Hsu. Query Processing in the SIMS Information Mediator, Advanced Planning Technology, A.Tate (ed), AAAI Press, 1996.
3. Badea L., Tilivea D. Intelligent Information Integration as a Constraint Handling Problem, Proc. of the Fifth International Conference on Flexible Query Answering Systems (FQAS-2002), October 27 - 29, 2002, Copenhagen, Springer Verlag, pp. 12-27.
4. Berger S., Bry F., Schaffert S., Wieser C. Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. Proceedings VLDB03, Berlin, September 2003, http://www.xcerpt.org/.
5. Bressan S., Goh C.H. Answering queries in context. In Proceedings of the International Conference on Flexible Query Answering Systems, FQAS-98, Roskilde, 1998.
6. Duschka O.M., Genesereth M.R.. Infomaster - An Information Integration Tool. Tool. Proc. International Workshop "Intelligent Information Integration", KI-97, Freiburg, 1997.
7. Decker S., Sintek M. 'Triple - an RDF query, inference, and transformation language', in Proc. of the 2002 International Semantic Web Conference (ISWC-2002).
8. Decker S., Brickley D., Saarela J., Angele J. A Query and Inference Service for RDF. QL'98 – The Query Languages Workshop, World Wide Web Consortium, 1998.
9. Fensel D., Angele J., Decker S., Erdmann M., Schnurr H.P., Staab S., Studer R., Witt A., On2broker: Semantic-based Access to Information Sources at the WWW, Proceedings of WebNet, 1999, pp. 366-371.
10. Fruewirth T. Theory and Practice of Constraint Handling Rules, JLP 37:95-138, 1998.
11. Garcia-Molina H., Papakonstantinou Y., Quass D., Rajaraman A., Sagiv Y., Ullman J., Vassalos V., Widom J. The TSIMMIS approach to mediation: Data models and Languages. In Journal of Intelligent Information Systems, 1997.
12. Kifer M., Lausen G., Wu J. Logical foundations of object-oriented and frame-based languages, Journal of the ACM, Volume 42 , Issue 4 (July 1995), pp. 741-843.
13. Kambhampati S., Lambrecht E., Nambiar U., Nie Z., Senthil G. Optimizing Recursive Information Gathering Plans in EMERAC. Journal of Intelligent Information Systems. Vol. 22, No. 2., March 2004, p. 119-153.
14. Kakas A., Kowalski R., Toni F. The role of abduction in logic programming, Handbook of logic in AI and LP 5, OUP 1998, 235-324.
15. Knoblock, C. et al. The ARIADNE Approach to Web-Based Information Integration, in: Intemational Journal of Cooperative Information Systems 10(1-2): pp. 145-169, 2001.
16. Levy A.Y. Logic-Based Techniques, in Data Integration Logic Based Artificial Intelligence, Jack Minker (ed). Kluwer, 2000.
17. Levy A.Y., Rajaraman A., Ordille J.J. Querying Heterogeneous Information Sources Using Source. Proc. 22nd VLDB Conference, Bombay, India. 1996.
18. Levy A.Y., Rousset M.C. Combining Horn Rules and Description Logics in CARIN. Artificial Intelligence Journal 104, September 1998.
19. Ludascher B., Himmeroder R., Lausen G., May W., Schlepphorst C. Managing Semistructured Data with FLORID: A Deductive Object-oriented Perspective. Information Systems, 23(8):589-613, 1998.
20. Marchiori M., Saarela J. Query+Metadata+Logic=Metalog, http://www.w3.org/TandS/QL/QL98/pp/metalog
21. Subrahmanian V.S. et al. HERMES: A heterogeneous reasoning and mediator system. http://www.cs.umd.edu/projects/hermes/overview/paper
22. Wiederhold G. Mediators in the architecture of future information systems, IEEE Comp. 25(3) 1992, 38-49.