Appears in *Proc. of the 3rd Int'l Conference on the Practical Application of Constraint Technology, held 23-25 April 1997 in London (U.K.), pp 35-46*

Signed Feature Constraint Solving

Jean-Marc Andreoli, Uwe M. Borghoff and Remo Pareschi
Rank Xerox Research Centre, Grenoble, France

Abstract

Intelligent brokering of information needs algorithmically tractable knowledge representations and corresponding constraint solvers to tackle the satisfiability problems involved. Feature constraints emerged soon as a convenient and elegant choice of knowledge representation. The full fragment of feature constraints, where sorts and features are allowed to be combined by all the logical connectives (conjunction, disjunction, negation and quantifiers), although very expressive, is hardly tractable. On the other hand, the subfragment called "basic feature constraints" (BFC), where negation and disjunction are simply forbidden, unneccesarily restricts applications such as knowledge brokers in an intolerable way.

The main contribution of this paper is twofold. On the one hand, it presents a fragment of feature constraints, called "signed feature constraints" (SFC), which allows limited use of negation, precisely capable of expressing the kind of operations needed during intelligent brokering, and on the other hand, discusses an efficient constraint solving method for SFC. Since early 1996, the SFC solver performs efficiently within the Constraint Based Knowledge Broker system (CBKB[1]). The CBKB system aims at heterogeneous information retrieval, schema integration, and knowledge fusion.

Key Words. signed feature constraints, constraint solver, entailment, intelligent brokering.

1 Background

Brokers are agents which can process knowledge search requests. Knowledge is taken here to be any piece of electronic information intended to be publicly accessible. Different, possibly distributed, information sources are assumed to be available, from a simple file in a user's directory to a database local to a site, up to a wide area information service (WAIS) on the internet, for example.

When receiving a request, a broker may have sufficient knowledge to process it, or may need to retrieve more knowledge. For that purpose, it releases sub-requests, aimed at other brokers. Thus, knowledge retrieval is achieved by the collaboration of all the brokers which are alternatively service providers processing requests and clients of these services generating sub-requests. We are not concerned here by the infra-structure required to support such collaboration, nor by the way knowledge is stored locally within each broker, but rather by the knowledge manipulations occuring within each broker.

In order to collaborate, the brokers must at least understand each other. This means that all the requests must be formulated in a common language (and also all the answers to the requests), even if the brokers may perform local translations. Logic provides the adequate language for such a purpose. A request can be expressed by a pair $\langle x, P\rangle$ where x is a logical variable and P a logical formula involving x, meaning "*Retrieve* knowledge objects x such that the property expressed by formula P holds". Interestingly, an answer to such a request can be expressed in the same formalism, i.e. a pair $\langle x, Q\rangle$, meaning "*There exists* a knowledge object x satisfying the property expressed by formula Q". The requirement here is that P must be a logical consequence of Q, so that the answer contains at least as much knowledge as the request. Moreover, the same logical formalism can be used to capture the scope of a broker, i.e. the area of knowledge it is concerned with: a broker with scope $\langle x, R\rangle$ means "*I am not capable of retrieving* knowledge objects x which do not satisfy the property expressed by formula R". In many situations, the scope of a broker may vary, because it gets specialized or, on the contrary, expands its capacities, either externally or due to the knowledge retrieval process itself.

In other words, logic provides a common language where both requests, answers and scopes can be expressed. Brokers then perform logical operations on these three components [2, 3]. The most important logical operation, from which all the others can be reconstructed, is satisfiability checking, i.e. deciding whether some object could satisfy

[1]A prototype of CBKB is available online: description and access from
`http://www.rxrc.xerox.com/research/ct/prototypes/cbkb/home.html`

the property expressed by a formula, or, on the contrary, whether it is intrinsically contradictory. Unfortunately, it is well known that this operation, for *full* Classical Logic, is not algorithmic, i.e. it is provably impossible to write a program which implements it and always terminates. Given this limitation, a lot of research in knowledge representation has been focused on identifying *fragments* of Classical Logic in which satisfiability is algorithmically decidable. The trade-off here is between expressive power and tractability: the empty fragment, for example, is obviously tractable, but it is not very expressive !... The most popular fragment which emerged from this research is known as "feature constraints", originally introduced in the context of linguistics [15] to represent syntactic information in Lexical Functional Grammars. The satisfiability problem in this case is also known as "feature constraint solving".
Traditionally, feature constraints are built from atomic constraints which are either sort or label constraints. A sort is a unary relation, expressing a property of a single entity. For example, `P:person` expresses that an entity `P` is of sort `person`. A label is a binary relation expressing a property linking two entities. For example, `P:employer->E` expresses that entity `P` has an employer, which is an entity `E`. Apart from sorts and labels, most feature systems also allow built-in relations such as equality and disequality.

2 Principles

Let S be the set of sorts (indexed by s) and L the set of labels (indexed by l). Let $\mathcal{V}$ be the set of logical variables (indexed by x, y, z). Atomic feature constraints (indexed by α) are defined by

$$\alpha \quad = \quad x : s \mid x : l \rightarrow x$$

Sorts and labels may not be semantically independent. For example, sorts can form a hierarchy, sometimes called the "is-a" hierarchy. Hierarchical links (e.g. a lion is a mammal) can be expressed by formulas of the form

$$\forall x \quad x : \texttt{lion} \supset x : \texttt{mammal}$$

Dependencies may involve not only sorts but also labels, as considered in U-Log [12]. In the rest of the paper, we assume that all the semantic dependencies can be formalized inside a theory $\mathcal{K}$, called the "knowledge theory", involving on the one hand atomic feature constraints and, on the other hand, some built-in predicates such as equality, disequality, ordering on numbers etc... Built-in constraints are indexed by β. Typically, we could have

$$\beta \quad = \quad x \dot{=} x \mid x \dot{\neq} x \mid x \dot{>} x \mid x \dot{\geq} x \mid x \dot{<} x \mid x \dot{\leq} x$$

but other constraints could as well be considered, such as the "substring" or "subword" constraints on strings, linear equations and inequations on finite domains or rational numbers, set constraints etc (as offerd by such systems as Prolog III [10], Eclipse [11], CHIP [9], ILOG Solver [14], Oz [13]).
Using these basic building blocks, we introduce three different fragments of classical logic, capturing, respectively, the requests a broker may receive, the answers it may produce, and the scope characterizing its current capabilities.

2.1 Answers

Concerning answers, we make the following two assumptions:

1. Each answer is fully characterized by a single knowledge object whose content is completely explicit for the broker. In other words, we do not consider here brokers which may produce partial or implicit answers.

2. The language of labels and sorts is expressive enough to directly express all the properties of a knowledge object.

With these assumptions, an answer from a broker can simply be expressed as a conjunction of atomic feature constraints. Formally, let's define the class ϕ^a of logical formulae

$$\phi^a \quad = \quad \alpha \mid \phi^a \wedge \phi^a$$

An answer is therefore a pair of the form $\langle x, \phi^a \rangle$ stating that there exists a knowledge object in the set $\{x \mathbin{/} \exists X \ \phi^a\}$, where X is the set of variables other than x occurring in ϕ^a. For example, an answer to a bibliographic query could be

```
X
  X : book
  X : title-> T          T : "The Moon and Six Pence"
  X : published-> P       P : 1919
  X : author-> A
      A : person
      A : name-> N        N : "W. Somerset Maugham"
      A : born-> B        B : 1874
```

The different constraints are here implicitly connected by a logical conjunction.

2.2 Requests

Concerning requests, the ideal language would be that consisting of all the logical formulae built from both atomic feature constraints and built-in constraints. However, this full fragment, although very expressive, is hardly tractable, and people have often considered a subfragment called "basic feature constraints" (BFC) where negation and disjunction are forbidden. Formally, this fragment is simply defined by the class

$$\phi \quad = \quad \alpha \mid \beta \mid \phi \wedge \phi$$

A request is therefore a pair of the form $\langle x, \phi \rangle$ triggering the search of all the available knowledge objects which are in the set $\{x \mathbin{/} \exists X \; \phi\}$, where X is the set of variables other than x occurring in ϕ. For example, a bibliographic query could be

```
X
  X : book
  X : published-> P       P < 1950
  X : author-> A
      A : person
      A : born-> B        P > B+40
```

which triggers a search for all the books published before 1950 by an author who was more than 40 at the time of publication.

2.3 Scopes

The language characterizing scopes could be restricted to that characterizing requests: indeed, the scope of a broker could be seen as the most general request it is capable of answering. However, completely disallowing negation in scopes (as in requests) puts strong limitations on the kind of operations a knowledge broker may wish to perform. In particular, we have identified a very common and powerful operation named "scope-splitting" [4], which relies on the use of negation. Indeed, a broker may wish to split its scope, specified by a pair $\langle x, P \rangle$ according to a criterion expressed by a formula F, thus creating two brokers with scope $P \wedge F$ and $P \wedge \neg F$. Thus, a broker in charge of various document information may wish to split its scope into two new scopes: "books written after 1950" and its complement, i.e. "books written before 1950 *or* documents which are *not* books"; this latter scope cannot be expressed using BFC, because negation and disjunction cannot be dispensed with. We have found that the scope splitting operation is needed in many situations, for example to implement brokers capable of memorizing and re-using information gathered during their lifetime. Semi-formally, if ψ represents the fragment characterizing scopes, the following properties should hold:

$$\begin{array}{lcl} \phi & \subset & \psi \\ \psi \wedge \phi & \subset & \psi \\ \psi \wedge \neg\phi & \subset & \psi \end{array}$$

stating that ψ contains BFCs and is stable by the scope splitting operations. This leads to

$$\psi \quad = \quad \phi \mid \psi \wedge \neg\phi$$

3 Signed Feature Constraints

A signed feature constraint (SFC) is a pair $\langle x, \psi \rangle$ where ψ is a formula belonging to the fragment identified above. The definition of ψ shows that each SFC is composed of a positive part and a list of negative parts, all of them being basic feature constraints. More formally, SFCs are characterized as follows:

Definition 1 *An signed feature constraint Ψ is a tuple $\langle x + \Phi_0 - \Phi_1 \ldots - \Phi_n \rangle$ where x is a variable, and $\Phi_0 \ldots \Phi_n$ are BFCs. Ψ is defined as follows:*

- *Let X_0 be the set of free variables other than x occurring in Φ_0;*
- *For each $k = 1, \ldots, n$, let X_k be the set of free variables other than x occurring in Φ_k but not in Φ_0;*

$$\Psi = \exists X_0 \; (\Phi_0 \wedge \bigwedge_{k=1}^{n} \neg(\exists X_k \; \Phi_k))$$

Notice that x is the only free variable of Ψ. It is called the constrained variable of Ψ.
For example, the following signed feature constraint

```
P
+ P : person
  P : employer-> E         E : "Xerox"
- P : nationality-> N      N : "American"
- P : spouse-> P'
      P': person
      P': employer-> E'    E' : "Xerox"
```

specifies a Xerox employee who is not American and is not married to another Xerox employee.

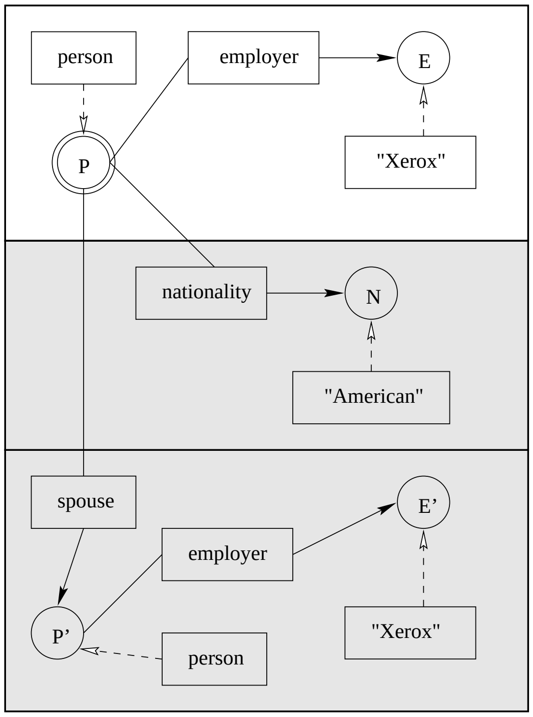

Figure 1: A Signed Feature Constraint (the negative parts are in grey)

We can represent it graphically as in Fig. 1. The round boxes denote the entities (logical variables), the sort relations (unary) are represented by dashed arrows labeled by the name of the sort in a square box, the feature relations (binary) are represented by plain arrows labeled by the name of the feature in a square box. The built-in predicates (not present in the example) are represented by rhombuses. The positive part of the SFC is contained in the top box and marks the distinguished entity of the scope (`P` in the example) by a double round box. The negative parts of the SFC are contained in the lower boxes in grey.
The stability of SFCs with respect to the scope splitting operation can be stated as follow:

> If the scope of a broker is represented by an SFC e_o, and this scope is split by a BFC e, then the two resulting split scopes e^+, e^- are both SFC.

Indeed, $e+$ is obtained by merging the positive part of e_o with the BFC e; and e^- is obtained by extending e_o with a new negative part containing e alone. For example, assume a broker in charge of a bibliographical database containing various documents (books, videos etc.) about Art, but not authored by an American. It is represented by the SFC

```
X
+ X : topic-> T          T : "Art"
- X : author-> A         A : nationality-> N       N : "American"
```

It may be split by the constraint "books published after 1950", expressed by the BFC

```
X
  X : book
  X : published-> Y      Y > 1950
```

The resulting scopes are simply

```
X
+ X : book
  X : topic-> T          T : "Art"
  X : published->Y       Y > 1950
- X : author-> A         A : nationality-> N       N : "American"
```

ie. "Art books published after 1950 but not by an american author" and

```
X
+ X : topic-> T          T : "Art"
- X : author-> A         A : nationality-> N       N : "American"
- X : book               X : published-> Y         Y > 1950
```

ie. "Documents about art, not authored by an American and other than books published after 1950".

4 Solving Signed Feature Constraints

Most constraint systems make a number of assumptions on the nature of sorts and features, formalizied in the knowledge theory $\mathcal{K}$. These axioms are crucial to the satisfiability algorithm, since they determine when a feature constraint is contradictory and when it is satisfiable.

4.1 Feature Axioms

For the purpose of simplicity, we make use here of a slight variant of the basic axiom system used in [1], although the principles of the method apply to other sets of axioms as well.

1. Features are functional: this means that if two pairs of entities which are constrained by the same feature have the same first term, they also have the same second term. For example, we may postulate that the feature `spouse` is functional (within a specific cultural setting), meaning that a person cannot have two spouses: if, for a person `X`, we have `X:spouse->Y` and `X:spouse->Z`, then the entities `Y` and `Z` coincide (ie. denote the same person). Other systems allow multi-valued features.

2. Sorts are disjoint: this means that no entity can be of two distinct sorts. For example, a book is not a person: we cannot have an entity `X` with `X:book` and `X:person`. Other systems consider hierarchies of sorts where some entities can have multiple sorts as long as they have a common denominator in the hierarchy.

3. There is a distinguished subset of sorts, called "value" sorts, so that no two distinct entities can be of the same value sort. Traditional basic elements (strings, numbers, etc.) are typical value sorts: for example, the string `"Xerox"` or the number `1950` are value sorts. Value sorts are not allowed to have features: this is the only axiom connecting sorts and features. Other systems consider more refined connections between sorts and features.

4. There is a distinguished built-in binary predicate, equality, with the traditional congruence axioms (which involve sorts and features). The axioms describing all the other built-in predicates are assumed to contain no mention of sorts and features.

Thus, the knowledge theory $\mathcal{K}$ consists of three sets of axioms.

Specific axioms for features and sorts :

Let τ, τ' denote any sorts, and f denote any feature.

$$\begin{array}{l}
\forall x, y, z \quad x \xrightarrow{f} y \wedge x \xrightarrow{f} z \supset y = z \\
\forall x \quad \neg(x : \tau \wedge x : \tau') \text{ if } \tau \neq \tau' \\
\forall x, y \quad x : \tau \wedge y : \tau \supset x = y \text{ if } \tau \text{ is a value sort} \\
\forall x, y \quad \neg(x : \tau \wedge x \xrightarrow{f} y) \text{ if } \tau \text{ is a value sort}
\end{array}$$

Congruence axioms for equality :

Let p denote any built-in predicate. The traditional congruence axioms are:

$$\begin{array}{l}
\forall x, y, z \quad x = x \quad \wedge \quad (x = y \supset y = x) \quad \wedge \quad (x = y \wedge y = z \supset x = z) \\
\forall x, y, z \quad (x \xrightarrow{f} y \wedge x = z \supset z \xrightarrow{f} y) \quad \wedge \quad (x \xrightarrow{f} y \wedge y = z \supset x \xrightarrow{f} z) \\
\forall x, y \quad x : \tau \wedge x = y \supset y : \tau \\
\forall \vec{x}, y \quad p(\vec{x}) \wedge x_i = y \supset p(\vec{y})
\end{array}$$

where i is some index in the list of variable $\vec{x}$ and $\vec{y}$ is identical to $\vec{x}$ except that $y_i = y$.

Built-in predicate axioms :

They must not mention sorts and features. For example, disequality can be axiomatized by

$$\begin{array}{l}
\forall x, y \quad x \neq y \vee x = y \\
\forall x \quad \neg(x \neq x)
\end{array}$$

Precedence constraints are axiomatized by

$$\begin{array}{l}
\forall x \quad \neg(x < x) \\
\forall x, y, z \quad x < y \wedge y < z \supset x < z
\end{array}$$

The built-in predicates $>, \leq, \geq$ can then be defined from $<$ and equality.

4.2 Constraint Satisfaction

Given the above axioms, we can now describe a constraint satisfaction algorithm. First, we assume that satisfiability over built-in constraints is decidable. This means that there is an algorithm which, given a formula F made of built-in constraints only, can decide whether F is a logical consequence of the theory $\mathcal{K}$ (written $\vdash_{\mathcal{K}} F$).

4.2.1 The BFC case

The constraint satisfaction algorithm for BFCs is defined by a set of conditional rewrite rules over BFCs (detailed below) which has the following properties

- *The system of rules is convergent and hence defines a "normal form" for BFCs.* This can be shown in a classical way by proving that the system is "Church-Rosser" (critical pairs converge) and "Noetherian" (the size of the terms strictly decrease by rewriting).
- *A BFC is satisfiable if and only if its normal form is not reduced to the contradiction.* The direct implication can be proved by showing that each rewrite step preserve satisfiability. The reverse implication can be proved by displaying a model which satisfies BFCs whose normal form is not reduced to the contradiction.

Thus the rewrite rules describe the steps of the constraint satisfaction algorithm. This algorithm always terminates because the system of rewrite rules is convergent. Notice that the definition of the rules rely on satisfiability tests of built-in constraints, which has been assumed decidable. This means that the algorithm is modular and can accomodate any kind of built-in constraints as long as a proper built-in constraint satisfaction algorithm is provided.
Describing a constraint satisfaction algorithm with rewrite rules is quite traditional. Rules can be implemented in a naive way is some symbolic language like Lisp or Prolog or be optimized, taking into account the properties of the specific built-in constraints which are used.
We represent a BFC as a pair $\langle B \mid \Gamma \rangle$ where B is a conjunction of built-in constraints and Γ a finite multiset of atomic feature constraints (also read conjunctively). $-$ denotes the contradiction. There are two sets of rewrite rules.

- The first set allows simplifications of the BFCs.

$$\begin{array}{lll}
\langle B \mid x \xrightarrow{f} y, z \xrightarrow{f} t, \Gamma\rangle & \mapsto & \langle B \wedge y = t \mid x \xrightarrow{f} y, \Gamma\rangle \text{ if } \vdash_{\mathcal{K}} B \supset x = z \\
\langle B \mid x : \tau, y : \tau, \Gamma\rangle & \mapsto & \langle B \mid x : \tau, \Gamma\rangle \text{ if } \vdash_{\mathcal{K}} B \supset x = y \text{ and } \tau \text{ is not a value sort} \\
\langle B \mid x : \tau, y : \tau, \Gamma\rangle & \mapsto & \langle B \wedge x = y \mid x : \tau, \Gamma\rangle \text{ if } \tau \text{ is a value sort}
\end{array}$$

- The second set of rules detects inconsistent (simplified) BFCs.

$$\begin{array}{lll}
\langle B \mid \Gamma\rangle & \mapsto & - \text{ if } \vdash_{\mathcal{K}} \neg B \\
\langle B \mid x : \tau, y : \tau', \Gamma\rangle & \mapsto & - \text{ if } \vdash_{\mathcal{K}} B \supset x = y \text{ and } \tau \neq \tau' \\
\langle B \mid x : \tau, y \xrightarrow{f} z, \Gamma\rangle & \mapsto & - \text{ if } \vdash_{\mathcal{K}} B \supset x = y \text{ and } \tau \text{ is a value sort}
\end{array}$$

The constraint satisfaction algorithm derived from these rules is based on the following property:

Theorem 1 *For any BFC consisting of the conjunction of built-in constraints B and a multiset of atomic feature constraints Γ,*

$$\langle B \mid \Gamma\rangle \quad \mapsto^{\star} \quad - \text{ if and only if } \vdash_{\mathcal{K}} \forall\neg(B \wedge \bigwedge_{c\in\Gamma} c)$$

4.2.2 The SFC case

The constraint satisfaction algorithm for SFCs can informally be described as follows. Given an SFC, its positive component is first normalized by the algorithm for BFCs. If the result is a contradiction, the whole SFC is unsatisfiable. Otherwise, the positive component (normalized) is inserted in each of the negative components, which are then normalized by the algorithm for BFCs. If a resulting negative component has a contradictory normal form, it is eliminated, and if it has a tautological normal form the whole SFC is unsatisfiable. The normal form for SFCs thus obtained has the following property:

An SFC is satisfiable if and only if its normal form is not reduced to the contradiction.

As in the previous case, the difficult part of the implication can be proved using model theory.
We represent an SFC as an *unordered* list of BFCs prefixed with a sign (+ or −); by definition, one and only one component is positive. Let $\mathcal{S}$ be an SFC. The SFC-normal form of $\mathcal{S}$ is written $\mathcal{S}^o$ and is obtained by the following algorithm:

```
Let $c_o$ be the BFC-normal form of the positive component of $\mathcal{S}$.
If $c_o = -$ Then
  Return $-$
Else
  $c_o$ is of the form $\langle B_o \mid \Gamma_o\rangle$
  Let $\{\langle B_i \mid \Gamma_i\rangle\}_{i=1,\dots,n}$ be the list of negative components of $\mathcal{S}$.
  Foreach $i = 1,\dots,n$
    Let $c_i$ be the BFC normal form of $\langle B_o \wedge B_i \mid \Gamma_o, \Gamma_i\rangle$.
  If there exists $i \in 1,\dots,n$ such that $c_i = \langle B \mid \Gamma\rangle$ and $\vdash_{\mathcal{K}} B$ and $\Gamma$ is empty Then
    Return $-$
  Else
    Let $I = \{i \in 1,\dots,n \text{ such that } c_i \neq -\}$
    Return $\{+c_o, \{-c_i\}_{i\in I}\}$
```

The following theorem justifies the algorithm.

Theorem 2 *For any SFC consisting of a positive BFC $\langle B_o|\Gamma_o\rangle$ and a list of negative BFCs $\{\langle B_i|\Gamma_i\rangle\}_{i=1}^n$*

$$[+\langle B_o \mid \Gamma_o\rangle, \{-\langle B_i \mid \Gamma_i\rangle\}_{i=1}^n]^o = - \text{ if and only if } \vdash_{\mathcal{K}} \forall\neg[(B_o \wedge \bigwedge_{c\in\Gamma_o} c) \wedge \bigwedge_{i=1}^{n} \neg(B_i \wedge \bigwedge_{c\in\Gamma_i} c)]$$

5 Field Application for Signed Feature Constraint Solving

This section describes the use of our SFC solver within the Constraint-Based Knowledge broker system (CBKB). A prototype of this system is available online at `http://www.rxrc.xerox.com/cbkb-cgi/main`. It requires a Java-enabled Web browser.

5.1 Information Gathering on the World Wide Web

CBKB, developed at RXRC Grenoble, realizes sophisticated facilities for efficient *information gathering* focusing on schema integration and knowledge fusion aspects. The current CBKB implementation consists of three kind of agents.

- Users, who input queries and process answers through a GUI [5].
- Wrappers, capable of interrogating heterogeneous information sources, which can provide answers to elementary queries (essentially various public bibliographic catalogues available on the Web, as well as preprint archives and an opera information repository) [7, 8].
- Brokers, which can manage complex queries (i.e. decompose a complex query, recompose the answers to subqueries, synthesize a full answer using the SFC solver) and which mediate between the GUI and the different wrappers [3, 6].

The core of the system is given by the brokers, which provide various important services such as intelligent cacheing, filtering and knowledge combination. Requests, results and brokers scopes are internally represented via Signed Feature Constraints (SFC). Requests correspond to partial specifications of the requested information. Results correspond to more refined specifications on the requested information (new constraints are added wrt the request). Our efficient SFC solver allows - apart from the scope splitting facility described in Section 2 - for local filtering / sifting of information.
CBKB-related projects in this area include: COIN (MIT), Harvest (Univ. Colorado), Garlic (IBM Almaden), HERMES (Univ. Maryland), Information Manifold (AT&T), InfoSleuth (MCC), KRAFT (Univ. Aberdeen / Wales / Liverpool, BT), ProFusion (http://www.designlab.ukans.edu/ProFusion.html), SIMS (ISI), and TSIMMIS (Stanford Univ.).

5.2 Dynamic Document Composition

Besides information retrieval from the World Wide Web, we are currently pursuing another possible CBKB application, namely *dynamic document composition* from on-line repositories: the widespread availability of new electronic sources of information such as E-mail, the Web, and other on-line information repositories multiplies also the number of electronic documents available. Indeed, documents can now be built dynamically by accessing and combining information existing over distributed sources. Again, symbolic constraints expressed as SFCs are at the core of this application domain.
Hierarchical mark-up languages like SGML can be used to define document templates that can be dynamically filled in with heterogeneous components. These documents can in turn be made permanent by storing them in document management systems, thus entering the normal document lifecycle.
Thus the CBKB project is also well-aligned with existing standardization efforts in document management and indeed provides an implementation of document search as specified by DMA (Document Management Alliance), an industry standard concerned with the search, retrieval, storage and conversion of documents on heterogeneous document management systems. DMA is the result of the merging of two previously existing standardization efforts: Shamrock (backed by IBM and Saros) and DEN (backed by Xerox and Novell).

6 Conclusion

We showed how, in an industrial research context, algorithmically tractable extensions to symbolic constraint solving techniques have been quickly adopted and applied with great success. In particular we presented a useful fragment of feature constraints, called Signed Feature Constraints (SFCs) together with a constraint solving method for SFCs.
Moreover, we illustrated a field application making use of SFC solvers, namely the Constraint-Based Knowledge Brokers (CBKB). The CBKB system's main target is information gathering over the World Wide Web. However, we also showed a way to use CBKB for the emerging market segment of dynamic document construction from geographically dispersed document repositories.

References

[1] H. Aït-Kaci, A. Podelski, and G. Smolka. A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science*, 122:263–283, 1994.

[2] J-M. Andreoli, U. Borghoff, and R. Pareschi. Constraint based knowledge brokers. In *Proc. of PASCO'94*, Linz, Austria, 1994.

[3] J-M. Andreoli, U. Borghoff, and R. Pareschi. The constraint-based knowledge broker model: Semantics, implementation and analysis. *Journal of Symbolic Computation*, 21(4):635–667, 1996.

[4] J-M. Andreoli, U. Borghoff, R. Pareschi, and J. Schlichter. Constraint agents for the information age. *Journal of Universal Computer Science*, 1(12):762–789, 1995.

[5] U. Borghoff, P.-Y. Chevalier, and J. Willamowski. Adaptive refinement of search patterns for distributed information gathering. In *Proc. of Int'l Conference EuroMedia/WEBTEC '96*, London, U.K., 1996.

[6] U. Borghoff, R. Pareschi, F. Arcelli, and F. Formatto. Constraint based protocols for distributed problem solving. *Science of Computer Programming*, 29, 1995.

[7] U. Borghoff, R. Pareschi, H. Karch, M. Nöhmeier, and J.H. Schlichter. Constraint-based information gathering for a network publication system. In *Proc. of 1st Int'l Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, London, U.K., 1996.

[8] U. Borghoff and J.H. Schlichter. On combining the knowledge of heterogeneous information repositories. *Journal of Universal Computer Science*, 2(7):515–532, 1996.

[9] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. The constraint logic programming language CHIP. In *Proc. of FGCS'88*, 1988.

[10] PrologIA Marseille (France). Prolog III. http://prologianet.univ-mrs.fr.

[11] European Computer Industry Research Centre Munich (Germany). Eclipse. http://www.ecrc.de/eclipse/eclipse.html.

[12] Paul Y. Gloess. U-Log, an ordered sorted logic with typed attributes. In *Proc. of the 3rd Int. Symposium on Programming Language Implementation and Logic Programming, PLILP91*, Passau, Germany, 1991.

[13] M. Henz, Smolka G., and J. Würtz. Object-oriented concurrent constraint programming in oz. In P. van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming*, pages 27–48. MIT Press, Cambridge, Ma, U.S.A., 1995.

[14] J-F. Puget and P. Albert. Solver: Constraints + objects = descriptions. In *Proc. of IJCAI'93*, 1993.

[15] W. Rounds and R. Kasper. A complete logical calculus for record structures representing linguistic information. In *Proc. of the 1st IEEE Symposium on Logic in Computer Science*, Boston, Ma, U.S.A., 1986.