Journal of Universal Computer Science, vol. 1, no. 12 (1995), 762-789
submitted: 11/12/95, accepted: 27/12/95, appeared: 28/12/95©Springer Pub. Co.

Constraint Agents for the Information Age

Jean-Marc Andreoli, Uwe M. Borghoff and Remo Pareschi
Rank Xerox Research Centre, Grenoble Laboratory
6, chemin de Maupertuis. F-38240 Meylan, France.
Email: {andreoli,borghoff,pareschi}@xerox.fr

Johann H. Schlichter
Institut für Informatik, Technische Universität München
D-80290 München, Germany.
Email: schlicht@informatik.tu-muenchen.de

Abstract: We propose constraints as the appropriate computational constructs for the design of agents with the task of selecting, merging and managing electronic information coming from such services as Internet access, digital libraries, E-mail, or on-line information repositories. Specifically, we introduce the framework of *Constraint-Based Knowledge Brokers*, which are concurrent agents that use so-called *signed feature constraints* to represent partially specified information and can flexibly cooperate in the management of distributed knowledge. We illustrate our approach by several examples, and we define application scenarios based on related technology such as Telescript and workflow management systems.

Key Words: multiagent coordination, agent-interaction, distributed problem solving, signed feature constraints, negotiation, cooperation strategies.
Category: H.3.3., I.2.

1 Introduction

New electronic sources of information, such as E-mail, Internet access and on-line information repositories flood the desktop environment of users with an evergrowing flow of information which, in order to be exploitable, demand efficient management. The inundation of electronic data coming from all kind of sources must convert into real knowledge in order to benefit the whole range of users from business people to casual surfers and shoppers on the Internet. Intelligent agents [CACM, 1994; Wooldridge and Jennings, 1995] interacting through multiagent systems have been proposed as the appropriate answer to this demand. Indeed, software processes of this kind may one day manage distributed knowledge by "living on the network" and manipulating electronic information on users' behalf.

A central question faces us in order to reach an effective deployment of such technology: How intelligent agents can be best designed and customized to meet users' individual information needs. The issue at stake concerns essentially one of adequate computational support. However, the motivations differ from those underlying linguistic frameworks for multiagent systems such as Actors [Agha, 1986] and Agent-oriented Programming (AOP) [Shoham, 1993] as well as of multiagent architectures, either "reactive" (see e.g. [Brooks, 1991]) or "deliberative" [Bratman et al., 1988] or "hybrid" [Kaelbling and Rosenschein, 1990]. In these cases the agents are assumed to be situated in an environment which they can modify while pursuing their own goals. These goals range from collecting empty cans, for simple robotic agents, or optimally solving scientific programming problems, for software agents in massively parallel multiprocessing architectures, to more complex types of activities for correspondingly more complex types of agents. Furthermore, the agents communicate either by passing messages (as in Actor languages) or by issuing, declining, or committing to requests, as well as performing other speech acts (as in the higher-level AOP languages). Thus, these agents explicitly communicate with their fellows and implicitly assume their situatedness in given environments. By contrast, agents primarily concerned with the elaboration and manipulation of information must have a more direct and explicit relationship with the environment since by its exploration they derive their very *raison d'etre*. Communication with fellow agents will at times be implicit and other times explicit: these agents effectively elaborate information and then communicate it, either to other agents or to humans. The recipients of information may be unknown to the senders – communication resembles more a radio broadcast or a conference presentation than a conversation between entities that know each other.

A computational model should satisfy a few precise requirements in order to support this notion of agency:

1. By definition these agents continually watch for information that meets preestablished criteria. For instance, in a given company, they can routinely scan news wires for breaking reports about the company's current customers, whoever they happen to be. Thus, it must be possible to express and implement agents' behavior in terms of a set of criteria through which information is filtered and selected. These criteria act as a partial specification of the information to come.
2. Electronic information domains are wide open lands where most of the times we do not know exactly what we are looking for, nor what we are going to find. In these conditions, a good way to guide our search is to explicitly exclude things we are not interested in. This can be conveniently expressed by freely mixing "positive" and "negative" requirements in the specification of the behavior of agents. Thus, users should be allowed to feed agents with such requests and criteria as "find me all books written by Umberto Eco which are not novels" or "I am not interested in reports on sales reps from Canada Customer Operations".
3. The scope of exploration of agents should be dynamically readjustable to optimize their work. As a minimal requirement, they should be capable of "focusing on targets," thus incrementally reducing their scope as they proceed. More intelligence could plausibly come from long-term memory, that is the remembrance of things past: They should be able to reuse the knowledge they have gained from executing a certain request in the context of other requests. Take for instance a request such as "find me all books by Umberto Eco which are not novels" and a subsequent request such as "find me all books by Umberto Eco which are literary essays."
4. We would also like to implement cooperative behavior of multiple agents on given tasks. Cooperation should arise naturally from handling queries involving the selection and composition of information from different knowledge repositories (often called backends) reachable through the Internet. Another example is the creation of compound documents on the fly from preexisting documents, according to some hierarchical description language

like SGML [ISO, 1986], with each part of the final document being assigned to a specific agent.
5. Finally, it should be possible to tune interagent communication in terms of different communication protocols according to such parameters as the nature of the problem to be solved, the underlying system architecture, etc.

In this paper we investigate these issues from the point of view of a computational construct that has already found widespread application in artificial intelligence and computer science, namely the notion of *constraint*. Constraints have been exploited mainly in the context of search and combinatorial optimization but their significance is more general and extends to the management and manipulation of information. In fact, constraints can be used to provide partial specifications on the possible values of variables. This paper illustrates how this capability can be exploited to implement predefined criteria for filtering and selecting information. The specific constraint framework we shall adopt is the Constraint-Based Knowledge Brokers (CBKBs) [Andreoli et al., 1994; Andreoli et al., to appear] model, which exploits constraints to support knowledge-intensive tasks executed by concurrent agents and views the management and manipulation of information in distributed environments as a form of distributed problem solving. A CBKB is capable of understanding and enacting both "requests" and "negations of requests," is self-sufficient in managing its own scope of exploration over a given information domain and is capable of knowledge reuse. Furthermore, different communication protocols for CBKBs have been defined that can be used to tune interagent communication and cooperation.

The remainder of this paper is organized as follows. In Sect. 2, we characterize the notion of multiagent interaction in the context of distributed problem solving. Agents are classified and given a set of general requirements. Agent cooperation is then illustrated in terms of CBKBs. Two different protocols for interagent communication are introduced and described, one supporting direct, explicit communication and the other supporting group-oriented communication. Sect. 3 introduces a specific type of constraints suitable for representing electronic information, namely signed feature constraints (SFC). In Sect. 4, SFCs are used to illustrate a number of specific issues of information management, such as interdependencies, thresholds, and reuse of information. Sect. 5 explains related scenarios. In particular we discuss negotiation in the contract-net protocol, Telescript as a promising agent infrastructure, and workflow management as an interesting application domain for remote programming. In Sect. 6, related work is discussed. Sect. 7 concludes the paper.

2 Multiagent Interaction

The area of Distributed Problem Solving (DPS) has led to various approaches which allow distributed (semi-)autonomous agents to cooperate in order to solve complex problems and accomplish tasks which might not be solvable by one individual system. From the problem solving point of view, distribution implies the decomposition of the problem into a set of subproblems and the dissemination of the subproblems to the appropriate agents which solve them autonomously and concurrently. The final solution of the global problem can be generated by composing the solutions of the subproblems. Thus, agents can be viewed as problem solvers which cooperate to generate the solution of the global problem.

2.1 Classification

We distinguish between passive and active agents. *Passive* agents act under direct user control. The user explicitly triggers the execution of agent functions, e.g. sorting and filing electronic messages in the user's mailbox. Unlike passive agents, *active* agents react to incoming messages, such as requests for information or the execution of functions, autonomously or semi-autonomously. *Autonomous* agents may perform actions without user involvement. They have enough knowledge about the problem domain and the contextual constraints to interpret received messages and react appropriately. During execution, the user has no direct control over the agent's behavior. On the other hand, *semi-autonomous* agents perform routine tasks for the user. Exceptional requests or situations are referred to the user who handles them personally. The behavior of semi-autonomous agents is directly controlled by the user who has read and write access to the rules which specify the agent's behavior.

Agents are used in a wide area of different application domains ranging from robotics, distributed sensoring to Computer-Supported Cooperative Work (CSCW) and information gathering [Wayner, 1994]. The emerging field of CSCW provides a demanding area for distributed problem solving. CSCW systems need to support the interaction of humans and overcome the difficulties of temporal and spatial distribution. For instance, agents may be used to support the scheduling of meetings (see [Sen and Durfee, 1991]). Another related application domain is that of workflow management and document processing where agents might be used to coordinate the tasks and information exchange between tasks and humans. The rapid growth of the Internet and the World-Wide Web have demonstrated the need for innovative and efficient ways of information gathering, and this provides the main focus for this paper. The World-Wide Web makes available an incredible amount of information; however, in many cases the user is unable to find and extract the desired information effectively. In this case agents may be used to collect relevant information, filter the search results according to contextual constraints, and present the resulting information to the user in an appropriate form. *Telescript* [White, 1994b] is an example of system providing infrastructural support for this type of agent application.

The *contract-net protocol* [Smith, 1980] was one of the first approaches to provide a general framework for DPS. It supports an application protocol for communication between problem solving agents and facilitates distributed control during the problem solving effort. Special emphasis is put on

- localizing those agents which are eligible for solving the created subproblems;
- the negotiation between agents for the information exchange with respect to subproblem descriptions, required agent capabilities and subproblem solutions [Davis and Smith, 1983].

The Rank Xerox Research Centre at Grenoble has developed the model of *Constraint-Based Knowledge Brokers* (CBKBs) which uses constraints to provide computational support for DPS. CBKBs explicitly separate aspects of local problem solving, based on computations specific to a single agent, from aspects of global problem solving, deriving from the interaction of different agents. CBKBs model active agents which act autonomously and concurrently. In the following sections some of the specific capabilities of the CBKB model will be discussed in more detail.

In order to effectively cooperate and participate in the problem solving effort, an agent must satisfy the following requirements:

- an agent must be able to communicate with other agents of the system (e.g. send and receive request/answer messages);
- an agent must be able to act upon receipt of messages.

2.2 Cooperation between Agents

The phenomenon of cooperation which is well-known in the human environment may also be applied to agent interaction. A number of different cooperation strategies between agents have been proposed, ranging from strongly hierarchical master-slave relationship, to the less hierarchical contract-net [Smith, 1980], to the sharing of common goals. In the latter case agents not only exchange information with respect to their individual tasks and problems, but they also communicate their goals. Thus, the agents follow shared goals when pursuing the problem solving activities.

In general, cooperation between agents is based on explicit communication, i.e. agents send messages to transfer knowledge and requests. The message content can range from values, formal and informal descriptions, to constraints. The CBKB model uses values and constraints to represent knowledge and requests for problem solving. The basic message types in the context of DPS are requests and answers. Usually messages are completely structured and are only intended for agent consumption; the messages are not in human-readable form. The message structures can be tailored to reduce network bandwidth and interpretation complexity by the agents. Both the contract-net protocol and the CBKB model apply structured messages to model agent interaction. An example for a system which uses semi-structured messages is *Object Lens* [Malone and Lai, 1988], which provides intelligent filtering and dissemination of electronic mail messages. Semi-structured messages are based on the notion of a semi-formal system [Malone, 1989] which:

- represents and interprets information that is formally specified,
- permits the human user to create and interpret formal information informally,
- allows the formal interpretation by the computer and the informal interpretation by the user to be easily changed.

Semi-formal systems are especially useful in heterogeneous environments where there is no clear separation between human tasks and agent tasks. They support the co-existence of humans and agents in the same environment. For example, some people use personal agents to cooperate in the distributed meeting scheduling process, while other people perform the required requests manually. Thus, semi-formal systems facilitate a smooth transition from a purely human-oriented environment to a completely agent-based environment. However, semi-formal systems use rather complex messages. This creates a significant network load and requires complex interpretation functionality by the agents.

2.3 CBKB Interaction Protocols

Within the CBKB model, two different agent interaction protocols have been designed:

- the request-subrequest protocol;
- the local caching protocol.

2.3.1 The Request-Subrequest Protocol

CBKB's request-subrequest protocol exploits dependencies between agents: the request carries an index that is added to all output information sent out as answers to the original request. In this way, requester and requestee are directly linked. Information is provided only if requested, and is sent only to the agents that have explicitly requested it.

The initial request carries an index which acts as an address for the requesting agents, as well as a description of the problem to be solved. A problem description in a request is instantiated as a constraint on the problem domain. Thus, a request is basically a constraint (with some additional information such as the index). An agent takes the problem description and simplifies it into subproblems. These descriptions of the subproblems are then submitted as subrequests in the same way as the initial request. The subrequests are individually indexed so that they can be collected into a solution by the requestee agents.

2.3.2 The Local-Caching Protocol

The local caching protocol does not link requesters with requestees. By contrast, as soon as a solution for a particular subproblem is available, it is broadcast to all existing agents. The initial request carries only a description of the problem to be solved; no index is associated with it. As before, an agent takes the problem description and simplifies it into subproblems. However, as a consequence of this protocol, for some of the subproblems solutions may already be known to the agents. The description of yet unsolved subproblems are then submitted as subrequests in the same way as the initial request, i.e. again without index. In this way, we obtain a situation of local *caching* of information for all existing agents, thus decreasing the overall amount of traffic, as we avoid the re-generation of the same requests from different requesters. On the other hand, we may end up storing information which never gets used.

2.3.3 Hybrid Schemes

Obviously, the two protocols above are at the very opposite ends of a spectrum of possible protocols and intermediate cases are possible, for instance, when automatic deliveries are done for subsets of agents. For many practical applications these cases seem to be the most useful, so we need techniques for assigning agents to appropriate "interest groups," and for allowing flexible tuning of group-based communication. Furthermore, there are cases where the best strategy for distributed problem solving may involve splitting the problem into subproblems which are optimally solved according to different protocols. Again, we need flexible ways for expressing such protocols, and for mixing them freely in the overall solution of a particular problem. Besides, we need ways of guessing the right protocol, or the right *melange* of protocols, for specific problems. This calls for contributions from such diverse fields as programming linguistics, learning and simulation.

3 Broker Agents and Constraints

In the CBKB model we formalize the problem-subproblem relationship which is at the basis of DPS via the notion of *generator*. Intuitively, a generator defines the decomposition of a given problem into subproblems and the composition of the subproblem solutions into the final solution of the problem. Operationally, generators are associated with a special kind of agents, the so-called *broker* agents. A broker incorporates the generator functionality together with the capability of dynamically spawning other agents (clones of the broker) for solving subproblems. The generator function g is implemented by applying input arguments to it and producing corresponding output information which represents the answers of the request to the broker.

3.1 Generator

Given an abstract domain of values $\mathcal{D}$, representing pieces of knowledge, a generator is a mapping $g : \mathcal{D}^n \mapsto \wp(\mathcal{D})$, which produces new pieces of knowledge from existing ones. The argument a_i, $i \in \{1, \ldots, n\}$ of the generator g represents a solution of the i-th subproblem. a_i may be either a value which was computed or retrieved from a database by the agent responsible for the subproblem, or a constraint. The number of arguments n of the generator g specifies the number of subproblems created by the broker out of the initial request; the broker has the arity n and is called *broker/n*. The arity n is only of local importance; it solely depends on the number of subproblems of the decomposed initial request. The sender of the initial request has no knowledge of the number of subproblems created by the broker/n. With respect to the decomposition of requests and composition of subanswers brokers act as autonomous agents. Thus, it is possible that a broker/n sends a subrequest to a broker/m where $n < m$. This approach to the hierarchical decomposition of requests and recomposition of answers exploits insights from deductive frameworks for parsing [Pereira and Warren, 1983] and database querying [Vielle, 1986].

In the current prototype implementation of the CBKB model the generator g needs solutions for all argument positions to apply its function, i.e. the appropriate agents must have provided solutions for the assigned subproblems. However, an extension of the current prototype is envisioned that incorporates more complex generators which also handle partial solutions, i.e. the solution of the global problem is generated from a subset of subproblem solutions. A simple example is the creation of a document which consists of several document parts. The generator g collects individual document parts and combines them into the final document. If document parts are not available g inserts automatically “missing subsection” into the final document.

As already mentioned earlier the generator g of a broker B composes answers to its request r out of the subanswers to subrequests. For an individual subrequest several contacted agents may provide multiple subanswers which return independently at broker B. Even multiple answers from a single agent may be received by the broker B at different times. At the receipt of a subanswer the generator g attempts to compose an answer to request r out of the newly received subanswer and the already previously received ones. The resulting answer will be checked by the broker B against the initial constraint of r in order to decide if it represents a valid answer. If there are multiple subanswers for certain subrequests available the generator g will construct all possible combinations to compose answers for the request r. Suppose the broker B decomposed the initial request r into two subrequests r_1 and r_2. For r_1 the broker B received two subanswers, and for r_2 three subanswers. The generator g will construct a solution space consisting of six potential solutions for r. By checking the initial constraints of r, only valid solutions are extracted from the solution space and propagated to the requester of r. In [Prasad et al., 1995], a related negotiation-driven multiagent retrieval approach is proposed where inconsistencies between different subanswers are dynamically resolved.

A set of generators identifies a class of subsets of the domain which are stable under these generators, that is, if the arguments a_i are within the subset, the knowledge generated by g is also within the same subset [Andreoli et al., 1994]. The class of stable sets is closed under intersection, so that it has a smallest element in the sense of inclusion, given by the intersection of all the stable sets. This minimal stable set, also called *minimal model*, represents the intended semantics of the set of generators.

3.2 Knowledge Representation

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 subrequests, 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 subrequests. We are not concerned here by the infrastructure required to support such collaboration, nor by the way knowledge is stored locally within each broker, but rather by the knowledge manipulations occurring 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. The most important logical operation, from which all the others can be reconstructed, is satisfiability checking, i.e. deciding whether some object could satisfy 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! A very popular fragment which emerged from this research is known as "feature constraints". The satisfiability problem in this case is also known as "feature constraint solving".

Traditionally, feature constraints are built from atomic constraints which are either sorts or features. 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 feature 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 features, most feature systems also allow built-in relations such as equality and disequality.

3.3 Constraints

The full fragment of feature constraints, where the atomic components mentioned above are allowed to be combined by all the logical connectives (conjunction, disjunction, negation and quantifiers), although very expressive, is hardly tractable. Therefore, we consider a subfragment, called "basic feature constraints" (BFC), where negation and disjunction are simply forbidden. Efficient constraint solving algorithms have been proposed for this subfragment. However, completely disallowing negation 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", 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 bibliographic information may wish to split its scope into two new scopes: "books written after 1950", which can be represented by the BFC

```
X
  X : book,
  X : year -> Y, Y > 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 reusing information gathered during their lifetime. Our approach presents on the one hand a fragment of feature constraints, called "signed feature constraints" (SFC), which allows limited use of negation, precisely capable of expressing the kind of scope splitting mentioned above, and on the other hand, an efficient constraint solving method for SFC.

3.3.1 Signed Feature Constraints

A signed feature constraint is composed of a positive part and a list of negative parts, both of them being basic feature constraints. 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. We can represent this SFC 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 main interest of SFC comes from the following property:

> 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"
```

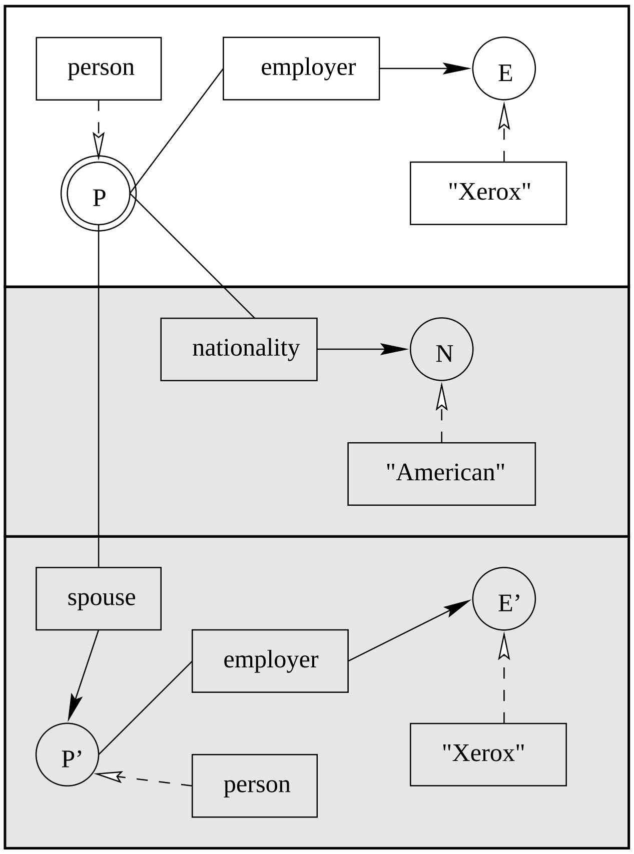

Figure 1: A signed feature constraint (the negative parts are in grey).

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

```
X
  X : book,
  X : year-> Y, Y > 1950
```

The resulting scopes are simply

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

i.e. "Art books written 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 : year-> Y, Y > 1950
```

i.e. "Art documents not authored by an American but not books subsequent to 1950".

3.3.2 Solving Signed Feature Constraints

Most constraint systems make a number of assumptions on the nature of sorts and features, called the axioms of the systems. These axioms are crucial to the satisfiability algorithm, since they determine when a feature constraint is contradictory and when it is satisfiable.

3.3.2.1 Feature Axioms

For the purpose of simplicity, we make use here of a slight variant of the basic axiom system used in [Aït-Kaci et al., 1994], 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 can consider 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 (i.e., 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.

These axioms form a theory $\mathcal{T}$.

3.3.2.2 Constraint Satisfaction

First, we assume that satisfiability over built-in predicates is decidable. This means that there is an algorithm which, given a formula F using only built-in predicates (F is also called a built-in constraint), can decide whether F is a logical consequence of the theory $\mathcal{T}$ (written $\vdash_{\mathcal{T}} F$).

Constraint satisfaction over BFCs is defined by a set of conditional rewrite rules over BFCs which have 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 a contradiction.* One implication can be proved by showing that rewrite steps preserve satisfiability. The reverse implication can be proved by displaying a model which satisfies BFCs whose normal form is not reduced to a contradiction.

Thus the rewrite rules describe the steps of the constraint satisfaction algorithm. The 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 our algorithm is modular and can accommodate any kind of built-in constraints as long as a proper built-in constraint satisfaction algorithm is provided.

Using rewrite rules for constraint satisfaction algorithm is quite traditional. They 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.

The algorithm for constraint satisfaction over 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 a contradiction.*

As in the previous case, the difficult part of the implication can be proved using model theory.

3.4 Implementation

The prototype implementation of the SFC solver is written in Prolog.

SFC are handled in a data structure `sfc(CV,POSFL,NEGFLL)` where `CV` represents the constraint variable, `POSFL` represents a list containing the feature entries for the positive part of SFC, and `NEGFLL` represents a list of lists where each list contains the feature entries for a single negative part of SFC. `NEGFLL` may be empty; then it is specified as empty list. In general, the SFC solver is realized as a list-transforming algorithm with additional checks for constraint satisfaction.

The built-in predicate for equations is solved using the unification mechanism and the build-in test "==" of Prolog. The precedence constraints ($>$,$<$,$\geq$, and $\leq$) are added through Prolog-predicates explicitly.

3.5 Searching the Backends and Caching of Information

Typically, at system initialization a set of initial brokers is provided. Each of these brokers has a predefined scope covering a subset of the domain of some backends. The sizes of the predefined scopes are application dependent and they may range from very specialized constraints to general descriptions of a constraint space covering the minimal model. By processing requests new brokers and agent specialists are cloned. Each of the newly cloned agents handles only a subset of their parent scope. This results in a continuous refinement of the scopes until the requested subset of the domain is handled by an agent specialist.

The scope of a broker is represented by a single SFC. There are labels $a_1, \ldots, a_n$ for each argument of the broker's generator. The label a_0 corresponds to the expected result.

Example 1. Assume a broker that is in charge of answering requests concerning opera information. A requester has to submit the name of a composer, and will receive all operas written by this composer in return. Assume further that the broker need not generate subproblems to generate a solution, i.e. the labels for $a_1, \ldots, a_n$ are not needed. Instead, in order to generate a solution it searches a backend, in our case, an attached opera database.

The initial scope of this broker is

```
X
+ X : a0-> O,
  O : opera
```

meaning that the broker is in charge of all possible operas (covered by the backend). The expected result is an opera. If the broker receives a request "Find me all operas of Richard Wagner", written as a BFC

```
O
  O : opera,
  O : composer-> C, C : "Richard Wagner"
```

it spawns an agent specialist in charge of exploring "Richard Wagner", and, as in the book example before, continues with an reduced scope.

As schematically illustrated in Fig. 2, the agent specialist in charge of exploring "Richard Wagner" searches an underlying opera database (or many databases where appropriate) and answers the request. The answer is an element of the domain and is represented by a BFC, e.g.

```
O
  O : opera,
  O : composer-> C, C : "Richard Wagner",
  O : name-> T, T : "Parsifal",
  O : number_of_acts-> NA, NA : 3
```

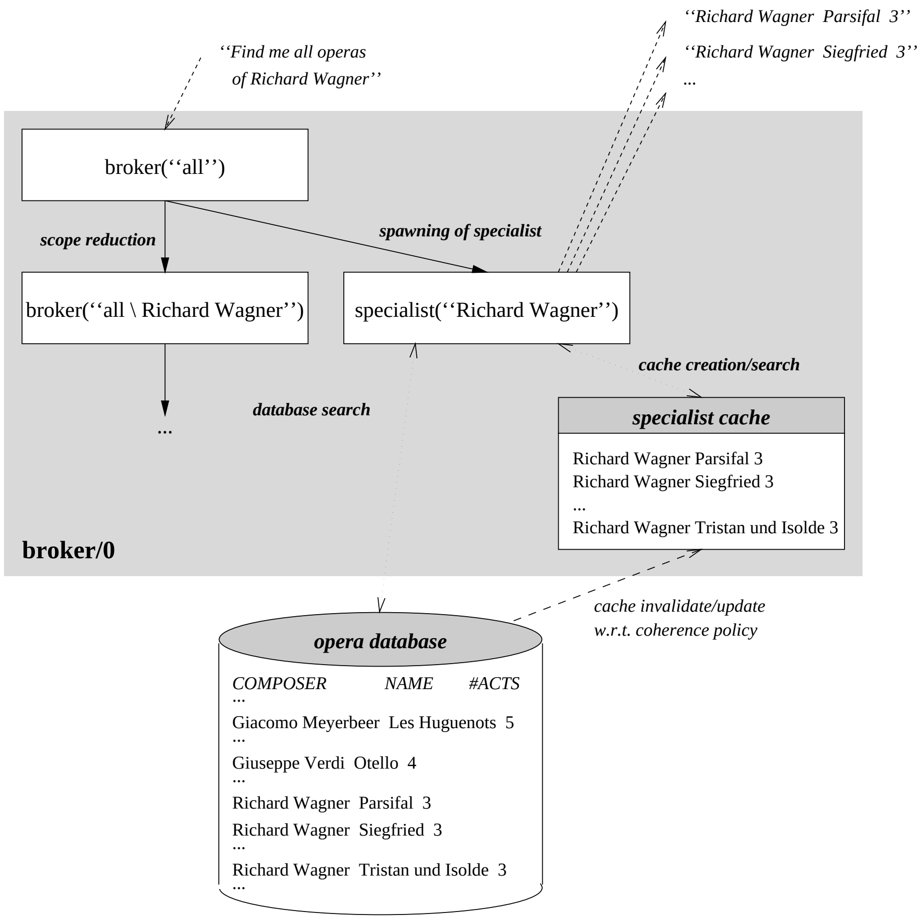

Figure 2: Specialist creation, caching and searching the backends.

The set of features "filled" by the answer depends on available information in the attached database.

This agent remains active as a specialist for Wagner operas, i.e., whenever another request for Wagner operas is sent, this specialist may simply return the results it has already collected. It is worth to point out that, due to its scope reduction, the parent broker will no longer react to requests concerning Wagner operas. On the other hand, a request "Find me all operas of Giuseppe Verdi with three acts" (provided the opera database has entries for act information), written as

```
O
  O : opera,
  O : composer-> C, C : "Giuseppe Verdi",
  O : number_of_acts-> NA, NA : 3
```

will lead to the spawning of a Verdi specialist for three-act-operas, and a corresponding (further) reduction of the broker's scope.

Now, we can show another characteristic of the approach: Imagine a follow-on request "Find me all operas of Giuseppe Verdi". This request will be answered by two specialists: First, the old Verdi specialists for three-act-operas will answer (typically, using already obtained information). Moreover, a newly created specialist for all Verdi operas not having three acts will answer. The requester will, due to earlier requests, get answers from two different specialists. Due to the scope splitting mechanism, redundant work is avoided. Already generated solutions for the overlapping part of the problem domain (three-act-operas of Verdi) are reused.

To make things complete, assume now a request "Find me all operas". This request will be answered by the three specialists already in existence. Moreover, a newly created, rather unconventional "specialist" for all operas but the ones by Wagner or Verdi (no matter whether three-acted or not) will answer. On the other hand, the scope reduction of the parent broker will lead to an empty scope. The parent broker will disappear.

The example raises some interesting questions. First, should a specialist send out its answers one by one, e.g. a single answer-message for each opera found, or should it collect all answers into a list, and send a single answer containing this list?

The second question concerns the valid reuse of information. The agent specialist remains active as a specialist for its subset of the constraint domain. Whenever, a request is sent concerning this domain, the specialist may reuse the already collected information, e.g. the list of Wagner operas. For Wagner or Verdi operas, this behavior seems appropriate. But assume a specialist for a living composer whose most important operas are still to be composed. We can see the specialist's domain as a cache, for which, of course, cache coherence policies are needed. As soon as cache-related information is updated in its original constraint store (e.g. updates in the underlying opera database concerning the specialist's composer), the cached information must either be invalidated or updated. Strategies concerning the valid reuse of information are discussed in Sect. 4.2.

4 Broker Processing

As we already discussed in previous sections the main tasks of an agent are the communication with other agents (e.g. sending and receiving of messages) and the reaction upon receipt of messages which may be either requests or answers to requests. The typical processing of a request submitted to a broker agent (broker/n) is – *cum grano salis* – accomplished through the following steps:

1. checking the problem description of the request with respect to the scope of the broker.
2. exploring the subset of the broker's scope that intersects with the constraint given in the problem description and spawning an agent specialist handling the subset currently under exploration. The broker itself continues to be in charge of the reduced scope which is derived from the "old" scope without the scope of the agent specialist.

3. applying back-dependencies to each of the broker's argument positions of the generator function g in order to simplify the problem description into subproblems.
4. checking the conditions to verify for which of the argument positions the simplified problem description can already be submitted as a subrequest (these conditions, which we call *threshold conditions*, will be discussed in more detail in Sect. 4.1).
5. submitting these subrequests in the same way as the initial request, i.e. with an index when using the request-subrequest protocol, or without when using local caching.
6. updating the (local) constraint store upon receipt of answers to these subrequests. Other argument positions of g may reach their threshold conditions and are submitted as in step 5.
   Once a combination of answers satisfies the initial request (after applying the broker's generator function), a solution is found. This solution is then sent to the initial requester, when using the request-subrequest protocol, or to all broker agents, when local caching is the protocol of choice. In addition to the scope splitting mechanisms where the creation of redundant agent specialists is avoided, local caching is of special interest when subproblems overlap. Redundant work is reduced by communicating relevant results in advance. As stated in [Oates et al., 1994], it is also interesting to see that a solution or even a partial solution generated by an agent might facilitate (by focusing or constraining) the problem solving of another agent. For instance, due to an "unsolicited" solution to a subproblem, a threshold could be satisfied and a (possibly more refined) subrequest could be launched.

Obviously, the steps described above simply illustrate a sort of upper-layer brokers providing solutions to a request. In another reading, we can see a broker/n as an agent that extends the functionality offered by lower-layer brokers. The solutions of a broker/n are higher-level, composite and tailored to the scopes of the lower-layer brokers. They reflect the assembled knowledge processed by the individual generators. However, someone has to provide real "basic" solutions. Synthesizing and combining answers is only possible when there are brokers in the CBKB model that need not further decompose the problem description into subproblems, but answer a request (immediately) by other means, e.g. by searching some backends. This is achieved by so-called brokers of an arity 0, written as *broker/0*. A typical example of a broker/0 might be an agent which handles queries to a database as shown in Example 1 or an agent that gets in contact with some service provider in the Internet.

A broker/0 reacts to an incoming request as in steps 1 and 2. However, instead of steps 3–6, the simplification of the problem description into subproblems, a broker/0

- searches/retrieves, e.g. by inspection of database files, or
- activates, e.g. by starting a calculation task within a spreadsheet application, or
- executes, e.g. by starting a process that evaluates some broker-internal sensoric data.

Of course, this is just a small fraction of possible activities a broker/0 may use to provide a solution to the request. It is clear that the set of all brokers having arity

0 forms the basis for searches over, most probably, heterogeneous data sources. A broker/0 also provides the interface to external tools and applications. Thus, the CBKB model can smoothly be integrated into an already existing application environment without changing legacy applications (see [Borghoff and Schlichter, 1995] for more details).

The following example illustrates the interaction of a broker/3 and some broker/0.

Example 2. Many people with an avocation for classic music may have asked the following question to extend their private library.

> "Find me all books, not written by a German author, where the title of the book is the name of a Wagner opera."

Writing this problem description as a feature constraint yields to the following BFC, constraining the request variable `R`:

```
R
  R : req_opera-> O,
  R : req_book-> B,
  R : req_person-> P,
  O : opera,
  O : name-> T,
  O : composer-> C, C : "Richard Wagner",
  B : book,
  B : title-> T,
  B : author-> PN,
  P : person,
  P : name-> PN,
  P : nationality-> N, N != N', N' : "German"
```

In order to answer the request, let a broker/3 compose results obtained from three different brokers, namely a broker/0 for operas, a broker/0 for books, and a broker/0 to verify the nationality of a person. The initial scope of broker/3 be

```
X
+ X : a0-> R,
  X : a1-> O,
  X : a2-> B,
  X : a3-> P,
  O : opera,
  B : book,
  P : person,
  R : req_opera-> O,
  R : req_book-> B,
  R : req_person-> P
```

The agent specialist cloned by broker/3 may decompose the problem domain into the following requests: First

> "Find me all operas of Richard Wagner".

This request may involve a first broker/0 that searches, for instance, a marketing server installed at Bayreuth. See Example 1 for more details.

Upon receipt of answers to this first request, the agent specialist extracts for every opera `O` the name `T` (e.g. Parsifal, Siegfried, Tristan und Isolde, etc.) and submits a second request of the form:

> "Find me all books with title `T`."

This request may involve a second broker/0 that executes a script to get in contact with a relevant service provider that may reside within the World-Wide Web. For example, at http://lcmarc.dra.com a form is provided to allow searches of the DRA-LCMARC database containing millions of relevant entries. If Telescript's visions [White, 1994a] become real it should also be possible to attach a Telescript engine to such a broker/0.

Upon receipt of answers to this second request, the agent specialist extracts for every book `B` the author `P` (e.g. for title "Parsifal", book authors are Piotr Bednarski, Friedrich Oberkogler, Hans-Jürgen Syberberg, Peter Vansittart, etc.) and submits a third request of the form:

> "Find me the nationality of the author `P`".

This request may involve a third *broker/0* that searches a commercial who's-who server to get the nationality of the author.

Upon receipt of an answer `N` (assuming that a person has only one nationality), the agent specialist feeds its generator with `O`, `B`, and `N` to generate a result that is sent back to the requester. The result generation is quite simple. If `N` is not German a solution is found, the answer `B`, i.e., the particular book, is provided.

Composed and/or verified using three different broker/0, the following would be a solution, and therefore be within the minimal model covered by CBKB:

> Vansittart, Peter. Parsifal : a novel / Peter Vansittart. London : P. Owen; Chester Springs, PA : U.S. distributor, Dufour Editions, 1988.

The final important aspect of broker processing discussed here refers to the life span of agents. As already mentioned above a set of initial broker agents is provided at system startup. By processing requests the system creates new agent specialists and modifies the scopes of its broker agents. The agents' life span is application dependent and may range from one individual user query to one user session to persistent existence. In the first case, agents are only created for handling the initial query. After the final answer has been generated all agents are removed. The reuse of cached information will be low. In the second case, agents live until the session is explicitly ended. Requests within sessions may lead to an increasing number of agents. Within one session the results of previous requests are reused to generate the answers of new requests. In the third case agents are persistent. Agents exist until they are explicitly removed from the system (e.g. at system shutdown, or attaching expiration times to agents). Thus, agents are similar to daemons in operating system environments. Again, agents reuse cached results of previous requests. However, because of the extended agent life span the cached information might be not up-to-date. Section 4.2 will discuss that aspect in more detail.

4.1 Interdependencies and Thresholds

The support of several interdependencies among subproblems models the general case of distributed processing, as required by DPS. Constraints provide a powerful and declarative approach to prune the search space of agents. Interdependencies may be used to model the order of sending the subrequests and thus the order of handling the subproblems. For example, the interdependency might specify that the subrequests are handled in sequential order, i.e. the subrequest k (argument position k of g) may sent only after the answers for the subrequest $k-1$ (argument position $k-1$ of g) have been received. Thus, interdependencies provide a powerful mechanism for modeling causal and temporal relationships between subproblems.

What we need are interdependency constraints and information thresholds.

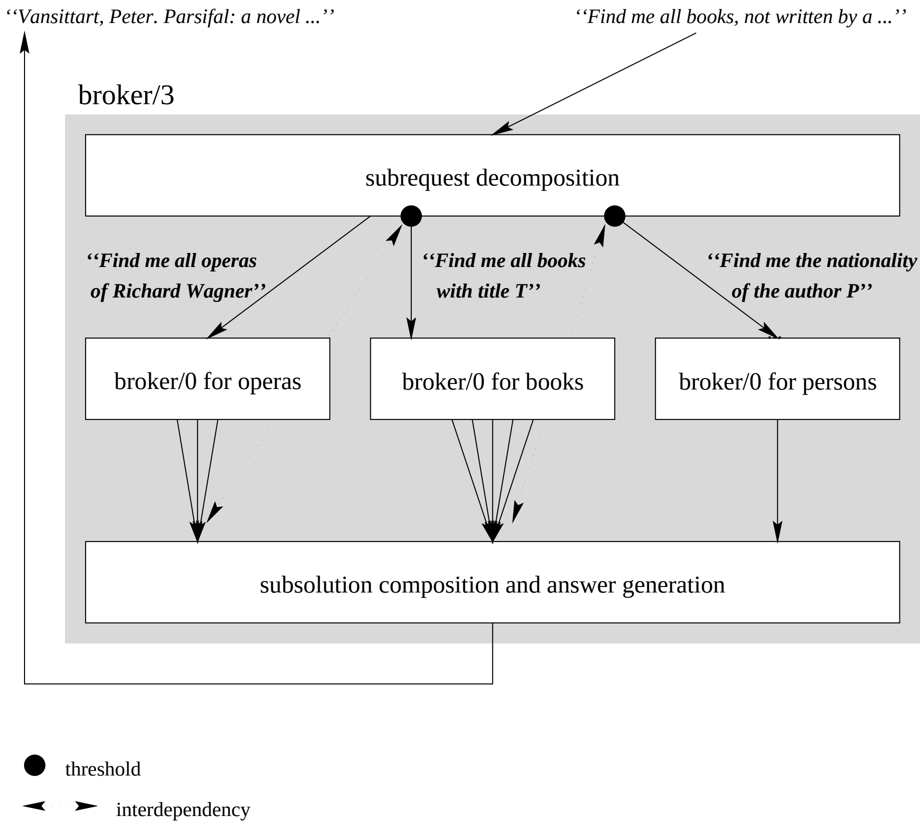

Figure 3: Broker interaction.

Interdependency constraints are quite simple. They are part of the constraint store. We have already used them in Example 2: Among other things, there is an interdependency constraint stating that the name of the opera and the title of the book must coincide, that is `O : name-> T` and `B : title-> T`, and another interdependency constraint forcing the coincidence of the author of the book and the person's name, that is `B : author-> PN` and `P : name-> PN`.

Information thresholds, on the other hand, are associated with each argument position. They are based on entailment tests, checking whether an argument entails a given (basic) feature constraint. Whenever a threshold condition of an argument position is satisfied the associated subrequest is triggered and sent to other brokers and agent specialists.

Example 3. In our example as illustrated in Fig. 3, it makes no sense to request a book without the knowledge of its title, or to query the nationality of someone who's name is not yet known.

Using the threshold mechanism and the interdependency constraint, broker/3 implements an argument ordering scheme, i.e., a request for argument position a_1 (Wagner operas) is sent first, a request for argument position a_2 (books with relevant title `T`) is sent whenever an answer for the first request arrives, i.e. whenever `nonvar(T)` (true if argument `T` is not a variable). Analogously, a request for argument position a_3 (nationality of a given author) depends on answers received for argument position a_2.

Thresholds may also be used to model repeated invocations of the same subrequest with refined input constraint values. Suppose the subrequest k was already sent and the associated answer has been received. After receiving the answer for another subrequest j the threshold for subrequest k is satisfied again; however, this time with new, refined constraint values. An example in document processing would be the following. Initially in the subrequest k an image of a person on a bridge is requested. The size of the image is constrained to one page. After performing the page layout subrequest the image size is constrained to a smaller size to fit at the appropriate place on the page. The subrequest k is sent again with the smaller size constraint.

4.2 Reuse of Information

Complex requests require interactions with many other agents and information stores in the network in order to gather the desired information. However, the execution of complex generator functions can be rather time consuming. Thus, in large information networks, such as the World-Wide Web the reuse of generated and already collected information is especially important. There are certain types of information which are quite stable. For example, the names and characteristics of known operas of Richard Wagner do not change; only the number of performances of them will change over time.

As a leitmotif, the interaction protocols of the CBKB model support the reuse of information and the avoidance of redundant generation of solutions. Actually, the constraint stores of agent specialists can be interpreted as caches of information. The cached information are values and constraints of already executed requests and subrequests to other agents. The quantity of reuse ranges from minimal to maximal. For minimal reuse all generated information is made potentially available, but it can be delivered only in response to specific requests (request-subrequest protocol). For maximal reuse all generated information is immediately broadcast to all agents (local caching protocol). The notion of quantity of reuse of information was formally introduced in [Arcelli et al., 1995]; it discusses heuristics in order to provide criterias for choosing between the two interaction protocols.

In static environments where knowledge does not change or is not extended, the information in the agent caches remains valid over time and can be reused by subsequent requests. On the other hand, in many applications the agents attached to a database have to deal with database entries that evolve over time due to updates. Older knowledge is replaced by newer knowledge or additional knowledge is defined and appended to existing database entries. These modifications must be propagated to the agent caches in order to provide reliable knowledge in the information network. Two different, basic cache coherence protocols are applicable:

- write-invalidate;
- write-update

In the first approach an information change invalidates all caches which utilize that information explicitly or implicitly, i.e. information derived from it by using generator functions. The constraint stores of the affected agents must be cleared. Thus, subsequent requests to an agent specialist with a cleared constraint store will require the renewed distribution of subrequests and the composition of all possible answers out of the newly received subanswers (see steps 5 and 6 for broker/n, or the activities of a broker/0 to interact with the external environment).

The write-update protocol propagates the modified information to all relevant agents which incorporate the new information into the already existing cache. The old cache entries must be identified and replaced. Additionally the execution of the generator function might be required and thus, new answers are generated which are sent to the requesters of the agents. The association between old and new cache entries might require an extension of the current CBKB model by assigning identifiers to constraints and values. It seems that the write-invalidate protocol fits better with the current architecture of the CBKB model.

The requirement of cache consistency is application dependent and we can identify three different application domains: First, domains where *no updates* occur at all and where *all processed information remains up-to-date*. Second, domains that are *update-tolerant*. For a given period of time, the application does not care about updates and reuses cached values even when already obsolete (e.g. statistics concerning the sales force where the latest sold disc player should not influence the results too much). In the World-Wide Web, location paths may be cached locally and cache entries are not updated automatically. If a user follows an invalid path then an error message is displayed (in some more user-friendly cases the path to the new web page location is displayed). Finally, there are the *update-critical* domain such as money brokering, tele-banking etc.

5 Related Technologies

In this section we try to explain how the proposed mechanisms of agent-based interaction can be applied to or be seen in related scenarios. In particular we focus on the contract-net protocol as a negotiation-based approach, Telescript as a promising agent infrastructure, and workflow management as an interesting application domain.

5.1 Contract-net

The contract-net protocol [Smith, 1980] can be seen as a particular instance of a CBKB-system with two broker roles, namely a manager and a set of bidders. The manager tries to localize a contractor among the bidders that solves a particular problem. Therefore the manager is equipped with a very specific generator for the bid selection.

The protocol is negotiation-based along five different phases: In the first phase, the manager announces the problem by sending a request for bids to the set of bidders, e.g. by sending a request constraining the problem domain and constraining the capabilities a potential contractor must have. In the next phase, the bidders check the problem constraints and propagate their bids, e.g. by tailoring the problem domain to the bidders' scope. The next phase comprises the selection of a contractor by the manager. Upon receipt of answers the manager invokes its generator. The result generation is quite simple. If the bid satisfies the initial constraint, a potential contractor is found. Among all potential contractors, a "best" (according to some problem dependent criteria) potential contractor is selected as contractor. In the fourth phase, the problem solving task is transmitted to the contractor. The final phase concludes with the problem solving itself.

5.2 Telescript

With General Magic's Telescript [White, 1994b] a promising technology is provided for the necessary infrastructure required to enable interacting autonomous agents in the so-called *electronic marketplace*. The electronic marketplace representing the Telescript world consists of a number of electronic places (e.g. a user's communicator, an electronic shopping center, etc.). Carrying a script to execute a particular task and permits that limit their capabilities, Telescript agents may migrate to perform transactions related to the visited places (e.g. a shopping order) or to simply search electronically provided information. The electronic places are homogeneous in the sense that they are able to interpret these scripts and to communicate with agents according to a defined protocol. In our proposed framework of CBKB, Telescript can play a vital role to fill the gap between the broker/n that synthesizes and combines answers received with respect to a complex request and the broker/0 that actually "gathers" information in the electronic marketplace. Obviously, a broker/0 could use a Telescript agent as its means to find and retrieve the information requested. On the other hand, a Telescript agent itself could internally implement CBKBs. In this case constraint handling, partial solution processing, and "agent specialist" creation would be added to the Telescript infrastructure. The latter issue corresponds to the Telescript ability the clone an agent (specialist) that does not migrate back to the requester in order to deliver the results but awaits further tasks within the place. One such task could be answering further requests using some locally cached information.

5.3 Workflow Management

The previous examples and application domains had a strong emphasis on information gathering as it is often integrated into the user environment. The user initiates a query by specifying the relevant feature constraints (e.g. in a form template); the query is transformed into a request which is then sent to brokers and agent specialists available in the system. However, the CBKB model may also be embedded in other application environments. Applications might directly communicate with brokers and agent specialists to gather and extract knowledge necessary for their internal functionality. In the following we will briefly discuss some of the possibilities how the CBKB model may be embedded into the workflow management domain.

In recent years there has been considerable work and publication related to workflow systems, models and studies. Also the growing interest in business process reengineering [Davenport, 1993] led to the development of commercial workflow systems [Abbott and Sarin, 1994] in order to support and improve business processes. McCarthy and Bluestein [McCarthy and Bluestein, 1991] define workflow management "as a proactive system which manages the flow of work among participants according to a defined procedure consisting of a number of tasks. It coordinates user and system participants, together with the appropriate data resources which may be accessible directly by the system or off-line, to achieve defined objectives by set deadlines." In the context of workflow management, agents and constraints may be used in a variety of different ways. Some of the possibilities are discussed below.

In general, a workflow consists of a task structure which models the tasks of the associated business process, and the temporal and causal interdependencies between tasks. Task structures are embedded into the organizational environment, and thus must incorporate organizational information. Examples for organizational information are the organizational structure (static information), information about people's work load or time schedules (dynamic information), information about organizational policies and strategies or other external and internal documents of organizational importance. During the specification and execution phase of task structures brokers and agent specialists might be used to gather the relevant organizational information and integrate it into task structures. For example, several brokers might access the static and dynamic information of the organization database and assign people to tasks according to their position in the organizational structure and their current work load. Additionally brokers could find and retrieve the relevant documents in the database in order to support the execution of individual tasks. In both examples, brokers and agent specialists are used for information gathering (see also example 2) to incorporate dynamically relevant information into task structures and thus support more efficient task execution.

Traditional workflow systems are inflexible with respect to exception handling and adapting to changing objectives. The goal-based workflow model [Ellis and Wainer, 1994] is an approach to improve that. In this model a workflow captures the goals of individual tasks and of the global business process in addition to the procedural steps. Goals and contextual information could be modeled as constraints. Brokers and agent specialists could be used during the planning phase to extract task structure templates and instantiate them appropriately to satisfy given goals and contextual constraints. Using the generator functionality, simple task structures could be composed into more complex tasks structures which incorporate interdependencies between tasks. The planning phase supported by brokers and the execution of tasks could take place intermittently to achieve a more reactive behavior with respect to changing goals and contextual constraints.

6 Related Work

Constraint-Based Knowledge Brokers (CBKBs) were first introduced in [Andreoli et al., 1994], where the request-subrequest protocol was also defined and described, together with such notions as reuse of information, recursion control, ordering of subrequest deliveries, and thresholds. In [Andreoli et al., to appear], the CBKB model is described in more detail. Complexity analysis are given concerning number of messages exchanged, number of agents cloned, and number of generations through the broker-attached generators. A simple example is provided in the domain of parsing of feature-based grammars.

In [Arcelli et al., 1995], the local caching protocol for CBKBs is introduced and compared with the request-subrequest protocol. Arguments are given, why, in many cases, it would be best to make use of both protocol schemes, or even devise hybrid schemes. Some examples, numerical values for the measure of reuse, and graphical illustrations of reuse potentials are provided.

For a conceptual characterization of Distributed Problem Solving see [Decker et al., 1989; Durfee et al., 1989]. However, so far DPS has been lacking a real computational model. Our contribution can be seen as step in the direction of providing a computational framework for DPS. More recently, a cooperative information gathering (IG) approach using a multiagent system based on DPS was illustrated in [Oates et al., 1994]. Additional relevant literature can be found in [Lander and Lesser, 1992].

We have mentioned Telescript [White, 1994b] as a possible solution for the infrastructural support of CBKBs on wide area networks. As for the implementation of the constraint solving aspects, a current prototype makes use of ForumTalk [Andreoli, 1995], a distributed language based on the *LO* model [Andreoli and Pareschi, 1991] for object-oriented rules-based computations. The built-in *LO* facilities for dynamic process spawning and broadcast communication are advantageously exploited in the definition of the agent interaction protocols. Other implementation choices are, however, possible [Borghoff, 1995]. A promising alternative is given by languages based on the Concurrent Constraint Programming model [Saraswat et al., 1991], such as Oz [Henz et al., 1995]. An advantage of these languages is that they have a built-in notion of constraint, and they come with *ask-tell* primitives for controlling the suspension/resumption of the activities of concurrent agents depending on the availability of relevant information. Thus, they provide a straightforward direction for the implementation of information thresholds for CBKBs.

7 Conclusion

In the course of this paper, we have shown how constraints can provide appropriate computational support for the management of electronic information, thus enabling a breed of intelligent agents that can help users to deal with the problem of "information overflow". The coming of age of a number of complementary technologies, e.g. programming languages for network navigation like Telescript, make this framework not just an elegant formal solution but also a promising practical approach.

References

[Abbott and Sarin, 1994] K. R. Abbott and S. K. Sarin. Experiences with workflow management: Issues for the next generation. In R. Furuta and C. Neuwirth, editors, Proc. 5th Int. Conf. on Computer-Supported Cooperative Work, pp. 113–120, Chapel Hill, NC, October 1994. New York: SIGCHI/SIGOIS ACM.

[Agha, 1986] G. A. Agha. Actors: A Model of Concurrent Computation in Distributed Systems. Cambridge, MA: MIT Press, 1986.

[Aït-Kaci et al., 1994] 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.

[Andreoli and Pareschi, 1991] J.-M. Andreoli and R. Pareschi. Communication as fair distribution of knowledge. In Proc. Conf. on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '91), pp. 212–229, Phoenix, AZ, November 1991. ACM SIGPLAN Notices **26**:11.

[Andreoli et al., 1994] J.-M. Andreoli, U. M. Borghoff, and R. Pareschi. Constraint-based knowledge brokers. In H. Hong, editor, Proc. 1st Int. Symp. on Parallel Symbolic Computation (PASCO '94), pp. 1–11, Hagenberg/Linz, Austria, September 1994. Lecture Notes Series in Computing **5**, Singapore, New Jersey, London, Hong Kong: World Scientific.

[Andreoli et al., to appear] J.-M. Andreoli, U. M. Borghoff, and R. Pareschi. The constraint-based knowledge broker model: Semantics, implementation and analysis. J. Symbolic Computation, to appear.

[Andreoli, 1995] J-M. Andreoli. Programming in forumtalk. Technical Report CT–003, Rank Xerox Research Centre, Grenoble Lab., France, 1995.

[Arcelli et al., 1995] F. Arcelli, U. M. Borghoff, F. Formato, and R. Pareschi. Tuning constraint-based communication in distributed problem solving. In Proc. 1st Int. Workshop on Concurrent Constraint Programming (CCP '95), Venice, Italy, May 1995.

[Borghoff and Schlichter, 1995] U. M. Borghoff and J. H. Schlichter. On combining the knowledge of heterogeneous information repositories. Technical report, Rank Xerox Research Centre, Grenoble Lab., France, September 1995.

[Borghoff, 1995] U. M. Borghoff. A procedural specification of the constraint-based knowledge broker model with partial results. Technical Report CT–004, Rank Xerox Research Centre, Grenoble Lab., France, May 1995.

[Bratman et al., 1988] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. Computational Intelligence, 4, 349–355, 1988.

[Brooks, 1991] R. A. Brooks. Intelligence without representation. Artificial Intelligence, 47, 139–159, 1991.

[CACM, 1994] **37**:7 CACM. Special issue on intelligent agents. Communications of the ACM, July 1994.

[Davenport, 1993] T. H. Davenport. Process Innovation: Reengineering Work through Information Technology. Boston, MA: Harvard Business School Press, 1993.

[Davis and Smith, 1983] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. Artificial Intelligence, 20, 1, 63–109, 1983.

[Decker et al., 1989] K. S. Decker, E. H. Durfee, and V. R. Lesser. The evaluation of research in cooperative distributed problem solving. In M. N. Huhns and L. Gasser, editors, Distributed Artificial Intelligence. Research Notes in Artificial Intelligence **2**. San Mateo: Pitman/Morgan Kaufmann, 1989.

[Durfee et al., 1989] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in cooperative distributed problem solving. IEEE Transactions on Knowledge and Data Engineering, 1, 1, 63–83, March 1989.

[Ellis and Wainer, 1994] C. A. Ellis and J. Wainer. Goal-based models of collaboration. Collaborative Computing, 1, 1, 61–86, March 1994.

[Henz et al., 1995] M. Henz, G. Smolka, and J. Würtz. Object-oriented concurrent constraint programming in oz. In P. v. Hentenryck and V. Saraswat, editors, Principles and Practice of Constraint Programming, pp. 27–48. Cambridge, MA: MIT Press, 1995.

[ISO, 1986] 8879 ISO. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). Int. Organization for Standardization: ISO IS, October 1986.

[Kaelbling and Rosenschein, 1990] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In P. Maes, editor, Designing Autonomous Agents, pp. 35–48. Cambridge, MA: MIT Press, 1990.

[Lander and Lesser, 1992] S. Lander and V. R. Lesser. Customizing distributed search among agents with heterogeneous knowledge. In T. W. Finin, C. K. Nicholas, and Y. Yesha, editors, Proc. 1st Int. Conf. on Information and Knowledge Management, Baltimore, MD, November 1992. Berlin, Heidelberg, New York: Springer-Verlag.

[Malone and Lai, 1988] T. W. Malone and K.-Y. Lai. Object lens: A spreadsheet for cooperative work. In Proc. 2nd Int. Conf. on Computer-Supported Cooperative Work, Portland, OR, September 1988. New York: SIGCHI/SIGOIS ACM.

[Malone, 1989] T. W. Malone. Semiformal systems and shared object spaces. In Proc. Groupware Technology Workshop, Palo Alto, CA, August 1989.

[McCarthy and Bluestein, 1991] J. C. McCarthy and W. M. Bluestein. The Computing Strategy Report: Workflow's Progress. Cambridge, MA: Forrester Research Inc., 1991.

[Oates et al., 1994] T. Oates, M. V. N. Prasad, and V. R. Lesser. Cooperative information gathering: A distributed problem solving approach. Technical Report TR-94-66, Dept. of Computer Science, Univ. of Massachusetts, Amherst, MA, 1994.

[Pereira and Warren, 1983] F. C. N. Pereira and D. H. D. Warren. Parsing as deduction. In Proc. 21st Annual Meeting of the Association for Computational Linguistics, MIT, Cambridge, MA, June 1983.

[Prasad et al., 1995] M. V. N. Prasad, V. R. Lesser, and S. Lander. Retrieval and reasoning in distributed case bases. Technical Report TR-95-27, Dept. of Computer Science, Univ. of Massachusetts, Amherst, MA, 1995.

[Saraswat et al., 1991] V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In Proc. 18th ACM SIGACT/SIGPLAN Annual Symp. on Principles of Programming Languages, pp. 333–352, Orlando, FL, January 1991. New York: ACM.

[Sen and Durfee, 1991] S. Sen and E. H. Durfee. A formal study of distributed meeting scheduling: Preliminary results. In Proc. Conf. on Organizational Computing Systems, pp. 55–68, Atlanta, GA, November 1991. New York: SIGOIS ACM.

[Shoham, 1993] Y. Shoham. Agent-oriented programming. Artificial Intelligence, 60, 1, 51–92, 1993.

[Smith, 1980] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C–29, 12, 1104–1113, December 1980.

[Vielle, 1986] L. Vielle. Recursive axioms in deductive databases: The query-subquery approach. In L. Kerschberg, editor, Proc. 1st Conf. on Expert Database Systems, Menlo Park, CA, April 1986. Benjamin/Cummings Publ. Company.

[Wayner, 1994] P. Wayner. Agents away. Byte, pp. 113–118, May 1994.

[White, 1994a] J. E. White. Telescript technology: Scenes from the electronic marketplace. General Magic White Paper, 1994.
[White, 1994b] J. E. White. Telescript technology: The foundation for the electronic marketplace. General Magic White Paper, 1994.
[Wooldridge and Jennings, 1995] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. Knowledge Engineering Review, 10, 2, 1995.

Acknowledgement

We like to thank Nathalie Glance for her careful reading of an earlier draft.

The work of the last author was done while visiting the Rank Xerox Research Centre at Grenoble on a sabbatical leave.