Finding and Moving Constraints in Cyberspace

P. M. D. Gray, K. Hui and **A. D. Preece**
Department of Computing Science
King's College
University of Aberdeen
Aberdeen AB24 3UE
Scotland
United Kingdom
{pgray|khui|apreece}@csd.abdn.ac.uk
http://www.csd.abdn.ac.uk/research/kraft.html

Abstract

Agent-based architectures are an effective method for constructing open, dynamic, distributed information systems. The KRAFT system exploits such an architecture, focusing on the exchange of information — in the form of constraints and data — among participating agents. The KRAFT approach is particularly well-suited to solving design and configuration problems, in which constraints and data are retrieved from agents representing customers and vendors on an extranet network, transformed to a common ontology, and processed by mediator agents. This paper describes the KRAFT system, discusses the issues involved in joining a KRAFT network from the point-of-view of information providers in Cyberspace, and examines the role of autonomous and mobile agents in KRAFT.

Introduction

Traditional distributed database systems provide uniform and transparent access to data objects across the network. These systems, however, are focused on the utilisation of data instead of other semantic knowledge stored in the database. One type of stored knowledge is constraints. Although traditionally used as database state restrictors, constraints as an abstraction constitute an important part of the semantics in a data model. We are interested in a *declarative* representation of *quantified* constraints which are self-contained abstract objects, because they can be used to represent domain-specific knowledge, partially solved solutions and intermediate results.

When used as mobile knowledge which is exported and attached to data, constraints restrict the way in which the data can be used and form relationships with other objects. This mobility, together with its declarativeness, allows constraints to be transported, transformed, combined and manipulated in a distributed environment.

Recently, it has been realised that constraints are a highly suitable representation for knowledge in distributed agent-based applications (Eaton, Freuder, & Wallace 1998), enabling novel approaches to the solution of design and configuration problems.

```
constrain each t in tutor
    such that astatus(t)="research"
no s in advises(t) has grade(s) =< 30;

constrain each r in residue to have
    distance(atom(r,"sg"),
      atom(disulphide(r),"sg")) < 3.7;
```

Figure 1: *The above examples demonstrate how Daplex/Colan (Bassiliades & Gray 1994) expresses a constraint on a university database containing student records. The same constraint language is applicable to the domain of protein structure modelling, as in the example restricting bond lengths.*

Background & Related Work

The KRAFT system (Gray *et al.* 1997) focuses on the utilisation and reuse of constraint knowledge held in distributed sources, by transforming it for use in various ontologies. Quantified constraints (Bassiliades & Gray 1994) (see figure 1) are a very general declarative form of predicate definition which can also compute using functions. They are effectively recipes for selecting or calculating things; they can be passed between agents, and may then be fused together or transformed into new recipes. This is the heart of the KRAFT architecture and it opens up many new possibilities.

Agent-based architectures are proving to be an effective approach to developing distributed information systems (Bayardo *et al.* 1997), as they support rich knowledge representations, meta-level reasoning about the content of on-line resources, and open environments in which resources join or leave a network dynamically (Wiederhold & Genesereth 1995). KRAFT employs such an agent-based architecture (Gray *et al.* 1997) to provide the required extensibility and adaptability in a dynamic distributed environment. Unlike most agent-based distributed information systems, however, KRAFT focuses on the exchange of *data and constraints* among agents in the system.

The design of the KRAFT architecture builds upon recent work in agent-based distributed information sys-

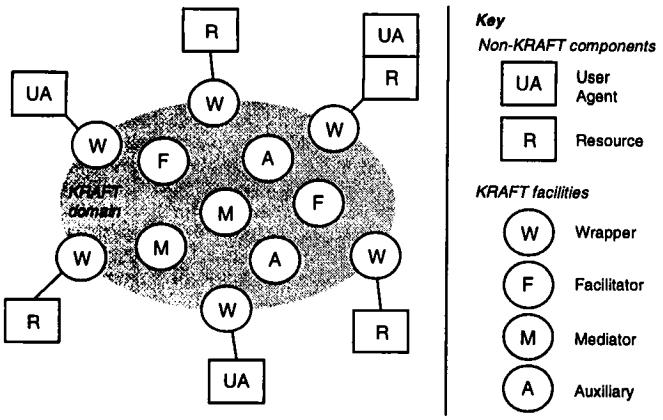

Figure 2: *A conceptual view of the KRAFT architecture.*

tems. In particular, the roles identified for KRAFT agents are similar to those in the InfoSleuth system (Bayardo *et al.* 1997); however, while InfoSleuth is primarily concerned with the retrieval of data objects, the focus of KRAFT is on the combination of data and constraints. KRAFT also builds upon the work of the Knowledge Sharing Effort (Neches *et al.* 1991), in that some of the facilitation and brokerage methods are employed, along with a subset of the 1997 KQML specification (Labrou 1996). Unlike the KSE work, however, which attempted to support agents communicating in a diverse range of knowledge representation languages (with attendant translational problems), KRAFT takes the view that constraints are a good compromise between expressivity and tractability. In its emphasis on constraints, KRAFT is similar to the Xerox Constraint Based Knowledge Brokers project (Andreoli, Borghoff, & Pareschi 1995); the difference is that KRAFT recognises the need to transform constraints when they are extracted from local resources, typically for reasons of ontological or schema mismatch (Gray *et al.* 1997) (Visser *et al.* 1997).

System Architecture

User agents and resources in KRAFT (figure 2) are interfaced to the KRAFT domain via wrappers. Wrappers provide translation services between the internal data representation (described in a local schema) of user agents and resources and the data representation (effectively an integration schema acting like a shared ontology) used within the KRAFT domain. If a resource is not sharable, or is incapable of handling asynchronous communication, then it is the responsibility of the wrapper to handle the necessary buffering and scheduling of requests to that resource. A KRAFT user agent serves the same purpose as an InfoSleuth (Bayardo *et al.* 1997) *User Agent* while a KRAFT wrapper is called a *Resource Agent* in the InfoSleuth terminology.

Facilitators provide internal routing services for messages within the KRAFT domain. They maintain directories of KRAFT facilities, their locations and what services they provide, and also details of their availability, load and reliability. Their principal function is to accept messages from other KRAFT facilities and route them appropriately. In particular, facilitators provide content-based routing services, so that they are able to route messages which are only partially addressed (or even wholly unaddressed) based upon the content of the message, or the service required. A KRAFT facilitator corresponds to an InfoSleuth *Broker Agent* and a KQML facilitator (Labrou 1996).

The primary focus in KRAFT is on "knowledge level mediation". In particular, we are aiming to provide a range of mediators which are specialised in the manipulation of knowledge in the form of constraints. Given the problem of deciding which catalogue to order a particular component from, we might first use a constraint extraction mediator to generate descriptions of the available components as conjunctions of constraints. These would then need to be translated into the domain ontology used by our design database, using one or more ontological mediators. The resulting collection of constraints must then be transformed into a single constraint representing the available options, and this must be fused into our existing set of design constraints. Finally, we ask one or more constraint solver mediators to coordinate the search for solutions to our problem, in a way that makes the best possible use of the available solving resources. KRAFT mediators are comparable to *Task Execution Agents* in InfoSleuth, and are an instance of Wiederhold's mediator concept (Wiederhold & Genesereth 1995).

Configuration Problems

One obvious area of application is in configuration problems. Traditionally these have been tackled by Rule-based Systems such as the famous XCON system, used for configuring VAX computers. Nowadays we tackle them more as Constraint Satisfaction problems. In the KRAFT architecture the domains of many of the variables will be entities stored in remote databases. Constraints on these entity types may be set by their makers, and stored with them in the database, as in the example given below. The constraints can then be found by a mediator and passed to a constraint solver together with other problem-specific restrictions. The solver then has to find feasible values to satisfy the constraints, as is common in engineering problems. However, note that the problem is complicated by constraints that refer to related instances of other entity types, whose values must be extracted from the database and checked for compatibility.

Usually configuration problems are solved by specially written pieces of software including pre-programmed constraints that take their parameter values from a number of flat files prepared by the designer. The KRAFT architecture generalises this to allow both the parameters and the constraints representing the problem to be searched for and selected and brought

together, over a network of nodes that may develop in various unanticipated ways. The agent architecture looks to be the best hope for coping with evolutionary change and the autonomy of different resource nodes.

For an example of the use of a KRAFT system, consider the problem of finding a number of parts that fit together to make something, or that work together in some way. Suppliers of these parts make catalogues, in the form of database tables, available over the Internet. However, the tables may have different semantics and hidden assumptions. These assumptions are often contained in an asterisked footnote or *small print* in the catalogue, for example: this part must be mounted in a housing of adequate size. Thus it is not enough just to make a distributed database query to find a list of possible parts; we must also ensure that these parts satisfy various constraints.

It is the knowledge in these constraints which we aim to reuse by transforming it to work in the context of a shared ontology that is being used to integrate the data. Thus we might have a constraint stored as metadata in the database for the AbComponents catalogue:

```
constrain each w in widget
to have width(housing(w)) >= width(w) + 5
    and width(housing(w)) =< width(w) + 15;
```

This constraint is expressed in the KRAFT Constraint Interchange Format (CIF), the first version of which is based on the CoLan language used to express semantics in the object database P/FDM (Embury & Gray 1995c). However, within the AbComponents database, the constraint might actually have been represented in some other form (as a trigger on a frame structure, for example) - it must be translated into a CIF constraint before it can be used by the KRAFT network. To make use of widgets from the AbComponents catalogue, we must translate this constraint into a form consistent with a shared ontology. This requires an understanding of the different terminologies used in the AbComponents database and the shared ontology:

```
constrain each w in wotsit
    such that source(w) = "AbComponents"
to have distance(left_neighbour(w),
        right_neighbour(w)) >= width(w) + 2
    and distance(left_neighbour(w),
        right_neighbour(w)) =< width(w) + 6;
```

There are various ways to use the transformed constraint; in a design, for example, it could be transformed and fused with another constraint on a particular usage of the widgets/wotsits as parts of containers:

```
constrain each c in container so that
    each p in parts(c) such that p is a wotsit
        and source(p) = "AbComponents"
has internal_diameter(c) >= width(p) + 2 and
    internal_diameter(c) =< width(p) + 6;
```

Alternatively we could represent the fused constraints as a collection of clauses in normal form. We can now use this fused information in various ways as explain later.

Roles of Constraints in KRAFT

A constraint is an excellent declarative way to specify domain-specific semantic features in a particular data model. It is an important abstraction which extends a data model in various ways so that it can address questions of importance today, in the era of the Internet.

We have chosen to use the Colan language developed for the P/FDM functional database system because it is based on Shipman's Daplex language (Shipman 1981), which is being used for its original purpose of integrating data expressed in different local databases using different local schemas. We have found this constraint language (figure 1) to be independent of the problem domain and able to represent the knowledge stored in a variety of local data models. It has the power of first order logic with safe expressions restricted by mixed quantifiers over finite domains of objects stored in databases or finite subranges of integers. It also has much of the power of a functional programming language for recursive computation.

Constraints in KRAFT come from various sources. The first kind of constraint is stored in a database in association with class descriptors for data objects, and it can be viewed as an attachment of instructions on how a data object should be used. They represent the *small print* conditions referred to earlier. Thus, data objects are annotated with declarative knowledge which can be transformed and processed.

These constraints are generic to all application problems that utilise the data. When a data object is retrieved, these attached instructions must also be extracted and satisfied to ensure that the data is properly used. Note that constraints are actually stored with a class descriptor but selection conditions in the constraint (e.g. `t in tutor such that name(t)="A. Smith"`) can be used to make it specific to a particular object.

The second type of constraint represents domain specific problem solving knowledge by imposing constraints on a solution database, which can be visualised as a database storing some or all the results that satisfy the application problem. These constraints are specific to the application problem and a potential candidate has to satisfy all of them in order to qualify as a solution. In its initial state, this solution database may not hold any actual data but provides a framework for specifying the problem solving knowledge. As solving proceeds, it gets populated with solutions that satisfy the constraints.

Constraints can also be used to represent restrictions placed by user specifications on the required solutions. Like other resources in the system, the user-agent serves as another information source feeding knowledge into the system in the form of constraint. User requirement constraints, in this way, are specific to an application problem instance of a user query.

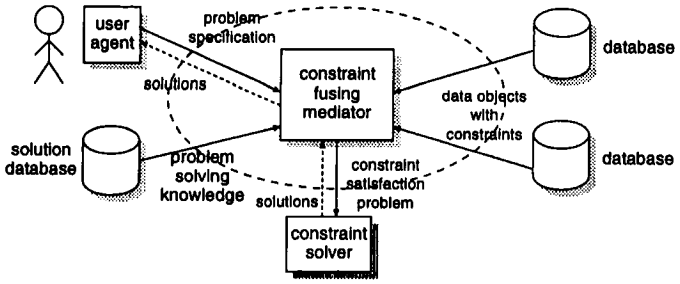

Figure 3: *This diagram shows how constraint fragments are exported from different resources and fused by the mediator to compose a constraint satisfaction problem.*

KRAFT Search & Fusion

The process of solving a design and configuration problem, therefore, is to retrieve data objects from other databases and populate the solution database while satisfying all the integrity constraints attached to the solution database and all relevant data objects.

With the role of constraints evolving from database state restrictors to portable predicates, a more efficient prune-and-search approach can be achieved by exporting constraint fragments to a constraint fusing mediator (figure 3) which composes the overall description as a constraint satisfaction problem (CSP) for a configuration task (Gray *et al.* 1998). The CSP is then analysed and decomposed into database queries and constraint logic programs which are fed across to distributed databases and constraint solvers, under the control of a mediator.

In general, there are three different ways to utilise the fused constraints, with increasing sophistication:

1. We can check the constraints against sets of objects retrieved by a distributed database query across the network, so as to reject any not satisfying the conditions.

2. We can use some combination of selection information in the constraint to refine the distributed database query, and thus do it more efficiently. This could also use the principles of semantic query optimisation.

3. To use constraint logic solving techniques (Embury & Gray 1995b); (Van Hentenryck 1989) to see if a complex set of interlocking constraints, whose form is not known until runtime, does have a solution.

Application problems in KRAFT are data-intensive CSPs as opposed to computation-intensive CSPs in many distributed constraint solving systems. A KRAFT CSP may involve simple constraints but a huge number of candidate data objects from multiple databases. This requires optimisation strategies that focus more on data filtering instead of computation efficiency.

A Constraint Fusing Example

To demonstrate constraint fusion from different sources, consider a configuration problem where a PC is built by combining components from vendors. The user specifies his requirement in the form of constraints through the user-agent. In this example, he specifies that the PC must use a "pentium2" processor but not the "win95" OS:

```
constrain each p in pc
    to have cpu(p)="pentium2"
    and name(has_os(p)) <> "win95"
```

For the components to fit together, they must satisfy certain constraints imposed by the solution database. For example, the size of the OS must be smaller or equal to the hard disk space for a proper installation:

```
constrain each p in pc
    to have
    size(has_os(p)) =< size(has_disk(p))
```

Now the candidate components from different vendors may have instructions attached to them as constraints. In the vendor database of operating systems, "winNT" requires a memory of at least 32 megabytes:

```
constrain each p in pc
    such that name(has_os(p))="winNT"
to have memory(p) >= 32
```

When we fuse all constraints together, we get the description of the overall CSP:

```
constrain each p in pc
    to have cpu(p)="pentium2"
    and name(has_os(p)) <> "win95"
    and size(has_os(p)) =< size(has_disk(p))
    and if name(has_os(p))="winNT"
        then memory(p)) >= 32 else true
```

Issues for Mobile Agents with Constraints

The KRAFT network architecture is being applied to the problem of gathering a specification for a configuration problem, including potential parts and their constraints. It is targeted at an *extranet* situation, where various vendors and potential customers form a virtual private network because they wish to share information. In order to do it they are prepared to conform (or map data and constraints) to an ontology that is shared but monotonically extensible (Gray *et al.* 1997).

Clearly, there is a cost associated with joining a KRAFT network (extranet), and in this section we consider the requirements imposed by the KRAFT approach on participating agents. There are two main issues to consider: (i) the need to conform to a *data model* and (ii) the role of agent *autonomy.*

Constraints need a Data Model

In many ways a data model makes up for the lack of full natural language comprehension by a computer since, if

we had this, agents would obviously exchange messages in natural language. What happens instead is that we partition possible data values and objects into distinct *data types*, and associate distinct real-world semantics with each, by saying what natural language concept best describes the partition or class(for example one given in the vocabulary of *WordNet*). We do the same with the named attributes and relationships applying to objects in each class. Now we have given the computer a means to check whether terms and concepts used in different databases are interchangeable (or just compatible), without fully understanding the meaning in a natural language sense. We validate all of this by appealing to a one-to-one correspondence between instances of objects and corresponding real-world objects as described in (Mylopoulos 1990).

We can also list which operations (or combinations of operations) on each data type are meaningful or not. This is typically done through a database schema, which is an important part of a data model. We may define the operations in relational algebra (for a relational model) or in terms of predicate logic and the lambda calculus, as in our functional approach. In fact in P/FDM we can define almost any computation, recursive or otherwise, on data structured according to an Entity-Relationship model including Subtypes; this kind of model is, of course, almost universally used in CASE tools for database design or software construction. Besides describing meaningful operations, we can also describe invariants preserved through state changes, which is the classic role for integrity constraints. This gives a more fine-grained level of semantics.

The point is that we now have a set of rules for calculating with and combining sets of data from heterogeneous sources, which produces well-formed results to which we can give a data type, and hence a label giving meaning. Thus the notion of *understanding* between humans has been replaced by a set of rules for operating on data, which knows when computations are meaningless without fully grasping their real-world meaning! We are, of course, familiar with this approach as used by type checkers in programming languages. However, as pointed out by (Brodie & Mylopoulos 1986) it is being used to represent the semantics of the external world, and not just those of the inside of a computing engine. Hence (Mylopoulos 1990) distinguishes ER models as having a different underlying ontology and semantics from those used in terminological or frame-based systems, despite a superficial resemblance.

Our work on constraints, both in KRAFT and previously (Embury & Gray 1995a), has convinced us that we need a data model in order to express constraints precisely. We need to establish finite domains of real world entities as a basis for quantification. We need to know what attributes or methods can be applied to each type of data (or entity). We need also to know about re-use (or overriding) of methods through specialisation of entity types. Only then can we state our constraints precisely. We need not, of course, include every aspect of each entity in our data model; we typically work within a restricted view of those aspects that are of interest.

Thus, where Internet web pages are just composed of natural language sentences, it is hard to know how to extract meaningful constraints automatically (or even just to check constraints), without making mistakes. The writer of the sentences may not even have kept to one consistent vocabulary, which makes the possibility for mismatches even greater. The only hope at present seems to be that particular subdialects of the new XML standard enforce the use of a named vocabulary or particular integration schema within certain sections of the document. Thus if, on a given web site, it is known that much of the textual information can be ignored and that one is only looking for a small number of facts in certain embedded tables or sections that are known to conform largely to a given data model, then this would be very useful. This is similar to the successful approach being followed by (Cohen 1998) in extracting information from HTML tables on web sites.

Use of constraints by autonomous agents

In the KRAFT system, as in many systems using mediators, agents reside on particular processors with specialised jobs such as facilitators, mediators, wrappers etc... Their location is found by using a facilitator. Thus, queries that involve collecting data from a number of locations can be performed by the mediator sending off a number of asynchronous requests. However, there then comes a synchronisation step where the information is put together and combined. If instead one was to follow the technique used to process distributed database queries, one would use the selection information implicit in the keys of tuples returned from one query to provide extra selection conditions embedded in queries sent to other databases, where the information from the databases is being joined conjunctively. Our approach in KRAFT is for the mediator to plan and generate the queries in cooperation with the constraint solver, based on the metadata it has about constraints (especially their number, type and selectivity).

In a system where agents were more autonomous, or even mobile, one can imagine setting off a number of agents in parallel with different strategies for visiting various sites and searching the data found there. It is rather like sending off a number of people on a treasure hunt, where they have to collect objects from several places, in no particular sequence. In this situation the agent is accumulating both data and constraints from various resources as the search progresses. The constraints act a bit like clues in the treasure hunt, in that they can cause the agent to modify its plan (similar to Conjunctive Query Optimisation (Smith & Genesereth 1985).

This is a promising strategy where we are interested in getting any one solution quickly. It is not suitable where we want systematically to compare all solutions

and find the optimum, or else to collect all the data and then use batch processing techniques on one machine, using economies of scale to improve performance. Much depends on whether the constraints are conjunctive or disjunctive. If the former, it pays to know what the most selective constraint is, even if the agent has to shop around. Otherwise you can get hooked on a large sub-problem and spend a lot of time collecting related data values, only to find they aren't needed because of another very restrictive selection condition! In this case the approach using planning should win out.

Again, if the search is for data satisfying disjunctive constraints, then it can be advantageous to have agents radio search results back to headquarters which can be broadcast to other agents to get them to abandon a search or else concentrate on a new lead!

Conclusions

The work reported here has concentrated on the collection, transformation and satisfaction of data intensive constraints, all associated with a data model describing stored relationships between objects in various distributed databases or knowledge bases. It is thus particularly relevant to *information-seeking agents.* We also emphasise that the domain of variables in the constraints is restricted either to subranges of integers or else to objects which belong to classes in object databases; usually those corresponding to entity types in an ER model.

The survey article on Agents and Constraints (Eaton, Freuder, & Wallace 1998) begins by noting a natural synergy between them, which we would strongly endorse. It uses the categorisation of (Nwana & Ndumu 1997) to classify the agents, and here we are particularly interested in *Information and Internet Constraint Agents.* We also concentrate on the use of generic agents which acquire constraints, of varying degrees of complexity, as they proceed to visit various sites, rather than the more usual task-specific agents that hold particular built-in parametrised constraints. This is an advantage we get from working within a functional model that allows us effectively to treat functions as data, and by using Prolog as our implementation language.

To date, we have concentrated on using the KRAFT architecture to collect constraints for solving configuration problems, which tend to be spatial in nature, rather than the temporal scheduling problems for which agents have been applied by (Liu & Sycara 1994) and others. KRAFT is not prohibited from collecting constraints for scheduling problems, but we have no experience at developing agents for use in this area, and we might very well use a different kind of solver.

Another unusual feature is that our architecture includes facilities for transforming constraints expressed in a local ontology so that they will conform to a shared ontology (with aspects of an integration schema) in order that the constraints may be expressed on a common basis and then combined. This is crucial for setting up a configuration problem. Thus our work with constraints is very dependent on the use of a local data model that can be mapped into an (extensible) shared ontology. This can be done easily enough in extranets, where those joining the network can agree to map their exported knowledge to the shared ontology. However, it is never going to be easy in the anarchy of the Internet, and the best hope seems to be agreement on the use of labels in XML to denote parts of web pages, often including tables of values, that conform to a named vocabulary or ontology.

We have speculated on the consequences of extending our architecture to use mobile agents. This has been addressed by (Andreoli *et al.* 1997); (Torrens, Weigel, & Faltings 1997). The interesting issues concern the tradeoff between agents behaving opportunistically, or else behaving in a more planned and coordinated fashion in which they send constraints back to a mediator and constraint solver that can help plan and coordinate a search for data to satisfy the constraints.

Many AI systems, such as Molgen (Stefik 1981) have studied moving constraints around within a single address space to help solve a complex planning problem. We have extended this to the propogation, collection and moving of constraints (particularly data-intensive ones) within an extranet, in order to set up a data-intensive CSP. This seems to be a very promising novel application for agents in Cyberspace!

Acknowledgements

We acknowledge support from BT for Kit Hui working on the KRAFT project, which is also supported by EPSRC. We would like to thank Graham Kemp (Aberdeen) and Zhan Cui (BT) and other KRAFT project partners for interesting discussions. Figure 2 and 3 in this paper are taken, with permission, from workshop proceedings (Wagner 1997) (Hui & Gray 1998).

References

Andreoli, J.; Borghoff, U.; Pareschi, R.; Bistarelli, S.; Montatnari, U.; and Rossi, F. 1997. Constraints and agents for a decentralized network infrastructure. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 39–44. Menlo Park, California, USA: AAAI Press.

Andreoli, J.-M.; Borghoff, U. M.; and Pareschi, R. 1995. Constraint agents for the information age. *Journal of Universal Computer Science* 1:762–789.

Bassiliades, N., and Gray, P. 1994. CoLan: a Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering* 14:203–249.

Bayardo, R. J.; Bohrer, W.; Brice, R.; Cichocki, A.; Fowler, J.; Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid, M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; and Woelk, D. 1997. Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of SIGMOD '97.*

Brodie, M. L., and Mylopoulos, J. 1986. Knowledge bases and databases semantics vs. computational theories of information. In Ariav, G., and Clifford, J., eds., *New Directions for Database Systems.* Ablex Publishing Corp.

Cohen, W. W. 1998. A web-based information system that reasons with structured collections of text. In Sycara, K. P., and Wooldridge, M., eds., *Proceedings of the Second International Conference on Autonomous Agents.* Minneapolis/St. Paul, MN USA: ACM Press.

Eaton, P. S.; Freuder, E. C.; and Wallace, R. J. 1998. Constraints and agents – confronting ignorance. *AI Magazine* 19(2):51–65.

Embury, S., and Gray, P. 1995a. Compiling a Declarative, High-Level Language for Semantic Integrity Constraints. In Meersman, R., and Mark, L., eds., *Proceedings of 6th IFIP TC-2 Working Conference on Data Semantics*, 188–226. Atlanta, USA: Chapman and Hall.

Embury, S., and Gray, P. 1995b. Planning Complex Updates to Satisfy Constraint Rules Using a Constraint Logic Search Engine. In Sellis, T., ed., *Proc. of 2nd Int. Workshop on Rules in Database Systems (RIDS '95)*, Lecture Notes in Computer Science 985, 230–244. Glyfada, Athens, Greece: Springer-Verlag.

Embury, S., and Gray, P. 1995c. The Declarative Expression of Semantic Integrity in a Database of Protein Structure. In Illaramendi, A., and Díaz, O., eds., *Data Management Systems: Proceedings of the Basque International Workshop on Information Technology (BIWIT 95)*, 216–224. San Sebastían, Spain: IEEE Computer Society Press.

Gray, P.; Preece, A.; Fiddian, N.; Gray, W.; Bench-Capon, T.; Shave, M.; Azarmi, N.; Wiegand, M.; Ashwell, M.; Beer, M.; Cui, Z.; Diaz, B.; S.M.Embury; K.Hui; A.C.Jones; D.M.Jones; G.J.L.Kemp; E.W.Lawson; K.Lunn; P.Marti; J.Shao; and P.R.S.Visser. 1997. KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases. In Wagner (1997), 682–691.

Gray, P.; Cui, Z.; Embury, S.; Gray, W.; Hui, K.; and Preece, A. 1998. An Agent-Based System for Handling Distributed Design Constraints. In Boddy, M., and Gini, M., eds., *Proceedings of Agents'98 Workshop on Agent-Based Manufacturing.* Minneapolis, USA: Dept. of Comp. Science and Eng., Univ. of Minnesota.

Hui, K., and Gray, P. M. D. 1998. Constraint and data fusion in a distributed information system. In Embury, S. M.; Fiddian, N. J.; Gray, W. A.; and Jones, A. C., eds., *Advances in Databases: Proceedings of 16th British National Conference on Databases (LNCS 1405)*, 181–182. Cardiff, Wales, U.K.: Springer Verlag.

Labrou, Y. 1996. *Semantics for an Agent Communication Language.* Ph.D. Dissertation, University of Maryland, Baltimore MD, USA.

Liu, J., and Sycara, K. 1994. Distributed problem solving throught coordination in a society of agents.

Mylopoulos, J. 1990. Object-Orientation and Knowledge Representation. In Meersman, R.; Kent, W.; and Khosla, S., eds., *Proceedings of the IFIP TC2/WG 2.6 Working Conference on Object-Oriented Databases: Analysis, Design & Construction (DS-4)*, 23–37. Windermere, U.K.: North-Holland (1991).

Neches, R.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senatir, T.; and Swartout, W. 1991. Enabling Technology for Knowledge Sharing. *AI Magazine* 12(3):36–56.

Nwana, H., and Ndumu, D. 1997. Introduction to agent technology. 1198:1–26.

Shipman, D. 1981. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems* 6(1):140–173.

Smith, D. E., and Genesereth, M. R. 1985. Ordering conjunctive queries. *Artificial Intelligence* 26:171–215.

Stefik, M. J. 1981. Planning with constraints. *Artificial Intelligence* 16(2):111–140.

Torrens, M.; Weigel, R.; and Faltings, B. 1997. Java constraint library: Bringing constraints technology on the internet using the java language. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 21–25. Menlo Park, California, USA: AAAI Press.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming.* MIT Press.

Visser, P. R. S.; Jones, D. M.; Bench-Capon, T. J. M.; and Shave, M. J. R. 1997. An analysis of ontology mismatches: Heterogeneity versus interoperability. In Fraquhar, A., and Grunninger, M., eds., *AAAI 1997 Spring Symposium on Ontological Engineering.*

Wagner, R., ed. 1997. *Proceedings of the Eighth International Workshop on Database and Expert Systems Applications.* Toulouse, France: IEEE Computer Society Press.

Wiederhold, G., and Genesereth, M. 1995. The Basis for Mediation. In Laufmann, S.; Spaccapietra, S.; and Yokoi, T., eds., *Proceedings of 3rd International Conference on Cooperative Information Systems (CoopIS-95).*