KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases*

P M D Gray†, A Preece†, N J Fiddian‡, W A Gray‡, T J M Bench-Capon#, M J R Shave#, N Azarmi+, M Wiegand+, M Ashwell‡, M Beer#, Z Cui+, B Diaz#, S M Embury†, K Hui†, A C Jones‡, D M Jones#, G J L Kemp†, E W Lawson‡, K Lunn#, P Marti‡, J Shao‡, P R S Visser#

† *Computing Science Department, University of Aberdeen*
‡ *Computer Science Department, University of Wales, Cardiff*
# *Computer Science Department, University of Liverpool*
+ *BT Laboratories, Martlesham Heath, Ipswich*

Abstract

The KRAFT project aims to investigate how a distributed architecture can support the transformation and reuse of a particular class of knowledge, namely constraints, and to fuse this knowledge so as to gain added value, by using it for constraint solving or data retrieval.

1. Introduction

Currently most people are accustomed to using the Internet and World-Wide-Web for browsing, mainly seeking technical and simple pictorial information, followed by the extraction of data for processing on their local computer. By this means, information can be gathered from many sources, but there is no systematic way to gain *added value* from the *combination* of information. We believe that the real challenge is to recognise and combine knowledge, in order to *enhance* the available information base, but *without* detailed human intervention. Thus the members of the KRAFT consortium[1] are working together to design and build a system having this capability. It will have intelligent mediators [22] that act as knowledge brokers and, more significantly, transform knowledge to make it useable by powerful problem-solvers at various sites on the network. The issue of knowledge transformation is crucial to the KRAFT project, since it makes possible a much greater degree of knowledge reuse [9, 10].

*The KRAFT Project is funded by grants from EPSRC (GR/K82765, GR/K82772, GR/K82789) and BT. For further information about KRAFT and related work, see: `http://www.csd.abdn.ac.uk/research/kraft.html`

[1] KRAFT = "Knowledge Reuse And Fusion/Transformation"

Much of the research into this area — which we will call Distributed Information Systems (DIS) — has concentrated on the distributed access and manipulation of *data*. While this is an important issue, with many problems remaining to be solved, it should be recognised that DIS must also manipulate and communicate *knowledge*. The manipulation of *structural* knowledge about data is now well understood and is the basis of most DBMS functionality, but knowledge also exists in other forms: integrity constraints, derivation rules, procedural knowledge, intensional query answers, production rules, classification hierarchies, and so on. All these types of knowledge can be reused within the wider context of DIS, provided that they can be liberated from the heterogeneous databases and knowledge bases in which they reside, and that appropriate tools for their manipulation and integration can be provided. This is the aim of the KRAFT project: to investigate how existing proposals for DIS architectures can support the transformation and reuse of a particular class of knowledge, namely *constraints*, and to propose further tools and architectural components for their manipulation and solution.

Currently there is great interest in DIS using mediators and facilitators following Wiederhold's original paper [22]. In particular, the DARPA-funded Knowledge Sharing Effort (KSE) [17] aims to facilitate sharing and reuse of knowledge bases. We wish to build on this work, but to explore particularly the ideas of knowledge reuse and functional transformation. Whereas the KSE has explored very general kinds of common-sense knowledge and natural language applications, we wish to look more specifically at those kinds of knowledge that can be represented declaratively as constraints, and transformed in various ways for use in design, scheduling, knowledge integration, and many other applications.

Thus we wish to extract constraint knowledge together with stored facts from various sites, and to combine it with constraints stated as a query or a design goal, so as to arrive at a more soluble problem. In other words, where various knowledge sources acting independently cannot solve a problem, we must enable them to combine information to solve it! Another point is that we need to learn how to use constraints, which are potentially much richer than tables of data as used currently in distributed databases. Although it is difficult to combine arbitrary pieces of procedural code taken from different databases, possibly in different languages, we hope to be more successful with constraints, because of their declarative form, whilst still retaining much of the utility and expressiveness of code.

1.1. Motivating Example

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 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 intend to reuse by transforming it to work in the context of a *shared ontology* (see Section 5) 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 [5]. 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 one of three ways:

- To 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.
- To 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 principles of semantic query optimisation [14].
- To use constraint logic solving techniques to see if a complex set of interlocking constraints, whose form is not known until run time, does have a solution [5, 19].

1.2. Transforming Constraints

We are in the early stages of manipulating and sending quantified database constraints as first class objects. One thing that is becoming clear is the subtle difference between the transformations used in supporting database views and those we have used in moving constraints from the context of one data model to another. In the case of views we are converting *data* items so that they *match* the specification expected by an application *program*. In KRAFT we are transforming quantified constraints (considered as a declarative form of *program*) so that they *match* the *data* in a common data model. Converting programs is always harder than converting data, but the declarative form of constraints makes it more tractable. Nevertheless it is a hard problem, needing an understanding of interoperability between differing ontologies.

The remainder of this paper is organised as follows: Section 2 describes the generic architecture of a KRAFT system, Sections 3 to 5 describe each of the components of the KRAFT architecture in more detail, and Section 6 concludes the paper.

2. The KRAFT Architecture

In keeping with the objectives of the KRAFT project described in the previous section, the KRAFT architecture is designed to support *knowledge fusion* from distributed, heterogeneous databases and knowledge bases. Components of the KRAFT architecture help applications and users to:

- locate data and knowledge relevant to their current needs;
- combine and refine data and knowledge in order to generate the *information* they require;
- identify and exploit the processing engines best able to solve their problems.

The KRAFT architecture will help information providers and system developers to:

- make their resources available to the widest population of users;
- cope with changes in the information or service they provide, while minimising the effects on client applications;
- keep account of users' access to their resources (e.g. for billing).

The basic philosophy of the KRAFT architecture is to define a "sanitized" communications space — a KRAFT domain — within which certain communication protocols and languages must be respected. Components which form part of the KRAFT architecture must conform to these protocols and languages, and must also provide sharable and asynchronous access to the services they provide. We call these components KRAFT *facilities*. Several important roles for KRAFT facilities have been identified to allow non-KRAFT components to be connected to KRAFT networks and to help manage communication between KRAFT components. Figure 1 shows a conceptual view of the architecture.

We will now describe the various roles that have been identified so far. As a starting point, we have taken three roles from other DIS research projects, namely *wrapping*, *facilitation* and *mediation*, but as the project progresses we expect to identify further useful roles, particularly in the area of knowledge manipulation.

Figure 1 also shows two types of component which are considered to be external to the KRAFT domain, but which must interface with it in some manner: *user agents* and *resources*. Users access the services of the KRAFT domain via *user agents*. A user agent may be a fully-fledged application such as a spreadsheet, or a simpler front-end such as a Web browser. Such user agents play the role of "consumers" of KRAFT services. *Resources* include both databases and knowledge bases; they also include constraint solvers and other kinds of processing engine. Such resources play the role of "service providers" to the KRAFT domain. They supply information (data or knowledge) in the form of responses to database queries, and solutions to constraint-satisfaction problems or KBS goals.

A user agent may be tightly coupled to a resource (as shown in the top-right of Figure 1); an example would be a KBS that is used directly by the local user to solve problems, while also being available on the network as a resource (for solving remote problems or as a source of knowledge for extraction). Such couples play the dual role of both consumer and service provider.

As we have said, user agents and resources are independent of the KRAFT architecture; they are interfaced to the KRAFT domain via *wrappers*. Wrappers provide translation services between the internal data formats of user agents and resources and the data format 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. In addition, wrappers must provide the high-level communication mechanisms needed to link the user agents and resources to the internal facilities of the KRAFT domain: the KRAFT facilitators and mediators.

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.

The final role that has been identified so far is that of *mediation*, which is the process of using domain knowledge to transform data in order to increase its information content. If this definition is rather general, this is because the term *mediation* actually encompasses a range of techniques for manipulating data sets in order to make them more useful. Examples of more specific kinds of mediation are:

- integration of data from multiple, possibly heterogeneous sources;
- summarisation of data sets in terms of trends, averages or exceptions;
- transformation of data from multiple sources to resolve mismatches of semantics, units of measurement

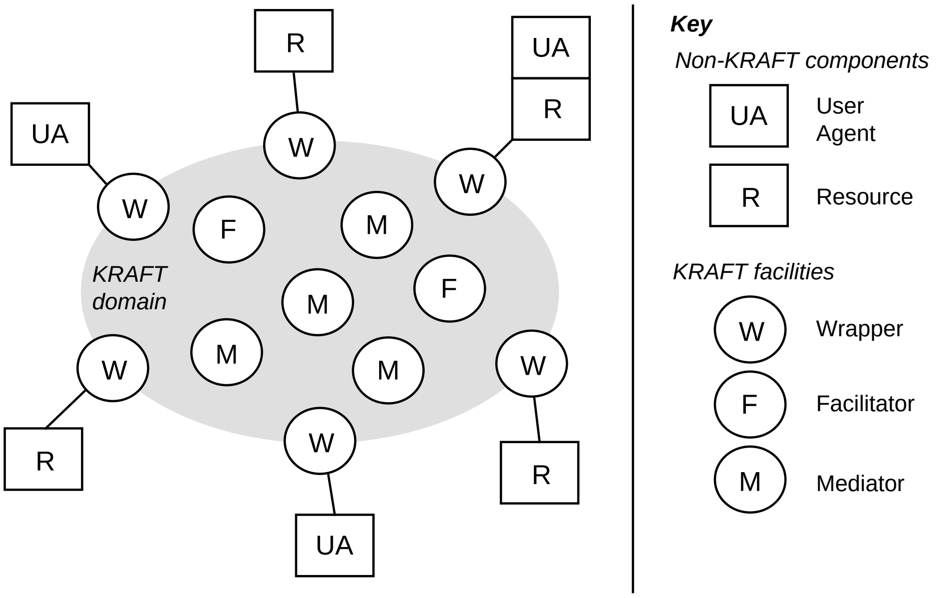

Figure 1. A conceptual view of the KRAFT architecture.

or granularity;

- filtering, sorting or clustering data according to relevance or some user-defined function;
- consistency checking and refinement of knowledge and data;
- maintenance of commonly-used sets of derived data.

Again, however, this is only a partial list, taken mainly from the existing literature (e.g. [22]), and we expect to be able to add further mediation roles specific to the manipulation of knowledge as the KRAFT project proceeds.

Unlike facilitators, *mediators* perform processing operations on messages within the KRAFT domain. They may combine messages, split messages, and transform the content of messages. The crucial point is that the operations performed by a mediator implement a particular task. The task may be highly specific to a particular application domain (such as a mediator for locating suppliers of *wotsits* that will fit into a particular *container*) but the mediator will be more generally useful if it performs a domain-independent task (such as locating suppliers of objects that satisfy a set of given constraints). However, it is also important that mediators provide a reliable, high-quality service, and therefore that they are small enough, and conceptually simple enough, to be easily maintainable.

One specific knowledge transformation role identified for mediators is the translation of knowledge expressed against one ontology (terminology) to the equivalent knowledge expressed against a different ontology. We call this role *ontological mediation*.

The fact that Figure 1 does not define concrete links between facilities within the KRAFT domain is significant. The communication relationship between all facility types is, in fact, many-to-many. In principle, any KRAFT facility can communicate with any other KRAFT facility. All KRAFT facilities communicate using common protocols and languages and, once one facility knows the network address of another, it can communicate directly with that facility. However, communication between wrappers will typically be conducted via facilitators and mediators, because user agent wrappers will not, in general, know which resources can provide the services they need at a particular time. It is also expected that mediators will make as much use of the content-based routing facilities provided by facilitators as is possible, as this will provide a degree of insulation from change in the availability or capabilities of network components.

3. Mediators in KRAFT

Mediation [21] is an architectural concept for large-scale distributed information systems which addresses the failings of the client-server architecture in respect of resilience to change, and reuse of code. It is now recognised that change is an integral feature of large-scale computer systems, and is not merely a result of poor system specification or implementation [23]. Computer systems must model the real-world, and the real-world is subject to change. Business strategies and management policies, governmental regulations, even the scientific or mathematical models underlying an application can all change over the lifetime of an application in ways that cannot be predicted during specification and implementation. A mediated architecture, as proposed by Wiederhold, improves upon more familiar models by promoting a finer granularity of code reuse, and by minimising the scope of changes and buffering existing components from their effects.

In Wiederhold's vision, a distributed information system consists of three layers (see Figure 2). In the top layer, we have the application programs (representing the needs of the various users of the DIS), while the lowest layer consists of the basic information sources (the databases, knowledge bases, simulation systems, forecasters, etc.) In between these, we have a layer of *mediators* — sharable software components which take data from the underlying sources and convert it into a more useful (and more usable) form for consumption by the application layer, or by other mediators. In this architecture, each mediator may extract data from one or more of the underlying information sources, and each application may make use of the transformed data provided by one or more mediators.

Wiederhold gives the following definition of a mediator: "A mediator is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications. It should be small and simple, so that it can be maintained by one expert, or at most, a small and coherent group of experts." [22] The crucial idea here is that a mediator *adds value* [24], in some sense, to the data retrieved from lower-level components. One of the commonest kinds of mediator is the "integrator mediator"[6, 16], which supports distributed querying by providing a coherent view of data from multiple heterogeneous sources. For example, a mediator might provide data about the employees of some company by integrating information stored in the personnel database and in the payroll record system. In this case, the knowledge encoded by the mediator is knowledge about the semantics of the data stored in each of these data sources. Other functions that a mediator can provide are abstraction of data to a common level of detail, conversion of units (e.g. imperial to metric) to allow comparison of data sets, statistical analyses and aggregation of data, and maintenance of commonly-used sets of derived data. All these functions serve to increase the "usefulness" of data—to make it easier for the user of that data (whether an application program or another mediator) to abstract from it the information that is needed.

It is clear that the mediator concept can facilitate software reuse, since commonly used services can be provided as autonomous, sharable mediator components that are accessed by as many applications as require that service. Those applications which make use of the data provided directly by the underlying information sources are completely unaffected by the provision of the new service. The three-layered architecture (which has obvious parallels with the ANSI/SPARC three-layer architecture [4] for database systems) also buffers the higher-layers against changes occurring in the lower layers. For example, if a change must be made to a data source used by some mediator, then a new version of that mediator can be provided which takes account of the change, while the old version of the mediator is modified to hide it. Then applications can either change to use the new version of the mediator, or can continue to use the old one. Wiederhold suggests an economic model for keeping the number of versions of mediators low, by increasing the charges for older versions to discourage their continued use [24]. Presumably other factors could also come into play, such as increased processing time, or known bugs which will not be fixed, to make older versions of mediators less attractive to users.

While we expect to be able to support all the mediation roles so far identified within the KRAFT architecture, our primary focus 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. Such manipulation will include:

- the extraction of relevant constraints from multiple underlying sources (i.e. distributed *meta-level* querying);
- translation of constraints expressed according to one ontology into constraints which can be understood by components which commit to a different ontology (*ontological mediation*);
- transformation and fusion of constraints according to the context of a particular problem;
- coordination of the available constraint solving resources for the solution of complex constraint satisfaction problems.

We can illustrate the use of these mediation roles by considering the design example given in Section 1.1. 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 disjunctive 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.

An important difference between Wiederhold's three-layer architecture and the KRAFT architecture is that, in

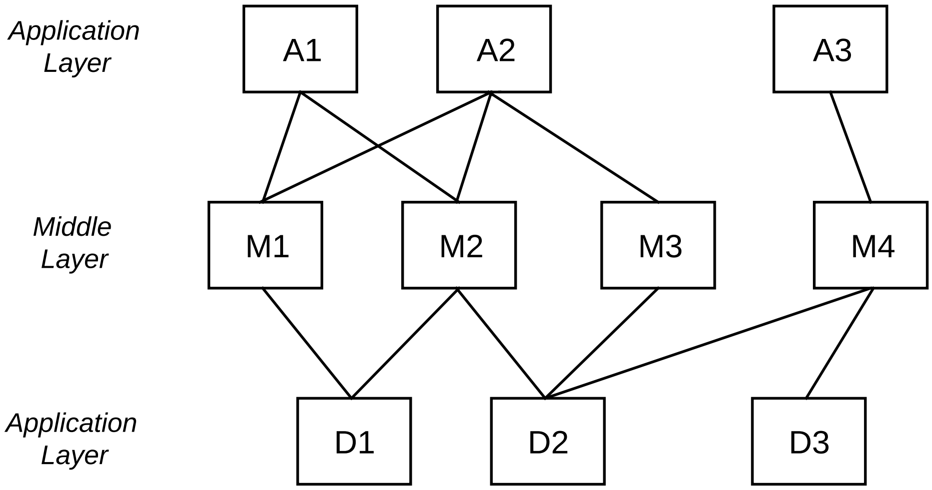

Figure 2. The Three Layer Architecture as envisaged by Wiederhold [22]

KRAFT, we do not distinguish so clearly between applications and information sources. This is because some software components may play both roles, in that some sources may initiate requests (e.g. to mediators) and some applications may be able to supply information for use by other processes. For example, the user of some knowledge base may require some mediated services in order to provide additional knowledge for solving their problem, while that same knowledge base may also contain knowledge that is more generally useful, and that may be retrieved by other components.

Another difference between the KRAFT architecture and the three-layer model is the idea that mediation is only one "role" for middleware components, and that "wrappers" and "facilitators" also have distinct roles. Wiederhold has tried to incorporate the facilitator concept into his middle layer by classifying it as a special kind of mediator [25] (since, for him, all middle layer components are mediators). We think it is better to keep the extremely useful roles of mediation and facilitation distinct. KRAFT is open to new classes of middleware facility, so that we do not have to weaken the concepts of mediation, facilitation and wrapping in order to accommodate middle layer components that don't quite fit into these specific roles.

4. Facilitators, Wrappers and Communication Language

In this section we will briefly describe facilitators, wrappers and communication languages which are used to bring heterogeneous resources together.

4.1. Facilitators

Facilitators are, like mediators, an important class of facility within the KRAFT domain. Their task, as mentioned in Section 2, is to provide internal routing services for messages within the KRAFT domain. More specifically, KRAFT facilitators will provide the following two main types of service:

Routing Services These services carry out message routing directly. That is, given a "request" from a user, a KRAFT facilitator will locate a set of "service providers" for that request. Typical routing services include a *yellow page service* which locates service providers based on a specific service type named in a user request, and a *content-based routing service* which locates service providers based on a user request without the help of a named service.

Routing Support Services These services do not locate "service providers" for a given user request. Rather, they provide support for routing services within the KRAFT domain. For example, *performance monitoring* is one of the routing support services which is used by routing services to decide which service provider should be used, in situations where a user needs one provider and there are several providers available. The selection criteria will include their respective reliability and their "cost". Other routing support services include subscription services and service provider browsing.

Note that facilitation has been used to describe a number of different roles in various projects [3, 15, 18]. In KSE, for example, facilitators perform the functions of problem decomposition, message routing and vocabulary translation, not just message routings as we do in KRAFT. However, we believe that having a specific facility within the KRAFT domain is advantageous. This is because it enhances "service provider independence". Without facilitators, any change made to any of the service providers, such as a change of physical address or a new provider signed onto KRAFT, will have to be propagated to all potential users of those services. Clearly, by using facilitators, such physical changes are made transparent to users. Users can thereby request services at a purely logical level.

Facilitators serve as the first contact point between the wrapper of a user agent (UA) and other facilities within the KRAFT domain, as shown in Figure 3. That is, a user request is always submitted to a facilitator first, via the UA wrapper. If the submitted request requires a specific named service, then a simple table look-up by the facilitator will suffice and will return the required physical address of that service. On the other hand, if a more general request is encountered, a facilitator will locate the relevant service(s) by carrying out some inference and analysis. It should be noted that this will potentially involve consulting ontologies. Note also that although facilitators are expected to perform some inference to locate service providers, they will not decompose a given request or integrate partial results. If a given problem needs to be decomposed before it can be solved, then KRAFT assumes that a mediator exists to do this. So the facilitator's task in this case will be to locate a relevant mediator. Of course, the mediator may use a facilitator again in the process of solving the problem delegated to it, as explained in Section 3.

Facilitators establish initial contacts between pairs of KRAFT facilities; once contact has been established, the facilitator "drops out" of the link, allowing the pair to communicate directly. The stages involved in this process are illustrated in Figure 3, where we have a wrapped user agent attempting to locate some service which, in this example, is offered by a mediator. The fact that this example features a wrapper contacting a mediator is not significant: the same interaction could occur between any two kinds of KRAFT facility. This explains why, within the KRAFT network, there may be many-to-many communication links between all kinds of facility.

4.2. Communication languages

In common with agent-based architectures [7], KRAFT needs both a language implementing the protocol of communicating agents and a language expressing the actual information to be extracted, reused and fused.

In KRAFT, the Common Command and Query Language (CCQL) is the external packaging of a message and is defined by a set of performatives based upon a subset of those used in KQML [7, 15] (e.g. `ask`, `tell`, `deny`, `error`, `register`).

The KRAFT Constraint Interchange Format (CIF) is the language carrying the actual information. In the current prototype version, the language used is CoLan [5], a constraint-based language, examples of which appear in Section 1.1. As CoLan was not developed for this particular purpose, it is being redesigned to ensure properties such as solution compactness and semi-decidability of logical entailment. It should be clear that a constraint language is an attractive approach when it comes to fusing knowledge because of the partial nature of the information involved [2].

4.3. Wrappers

The above two languages are to be the *Lingua Franca* and will be used universally within the KRAFT domain. To achieve this, each external component that does not comply with CCQL and CIF has to be "wrapped" by a dedicated component, a *wrapper*. The issues involved in wrapping vary greatly depending on the resources to be wrapped. As we are currently using P/FDM databases as resources, the wrapping procedure is syntactically straightforward; other resources will be the subject of on-going research [1]. However, there are two major issues that a wrapper needs to address besides mere syntactical translation. One issue is the communication model supported: CCQL performatives need to be mapped to requests that the resource is able to fulfil. The other issue is the ontological problem: there may be significant mis-matches between the ontology underlying a message and that underlying a given resource. This is the subject of the next section.

5. Ontologies in KRAFT

Sharing knowledge between resources requires not only agreement on protocols and formats as described above: it also requires agreement on the specification of the content of the knowledge to be shared. Any formalised body of knowledge will embody, implicitly if not explicitly, a conceptualisation of the domain with which its knowledge is concerned. By a "conceptualisation" we understand "the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships which hold between them" [8]. Two resources can share knowledge only to the extent that they share the conceptualisation: in the extreme case, if one resource has no concept of a particular kind of object, knowledge about such objects will be meaningless to it.

Of course, it would be possible simply to assume — or hope — that the conceptualisations are sufficiently in harmony to permit knowledge sharing. But any such assumption would be dangerous. In order to have any guarantee

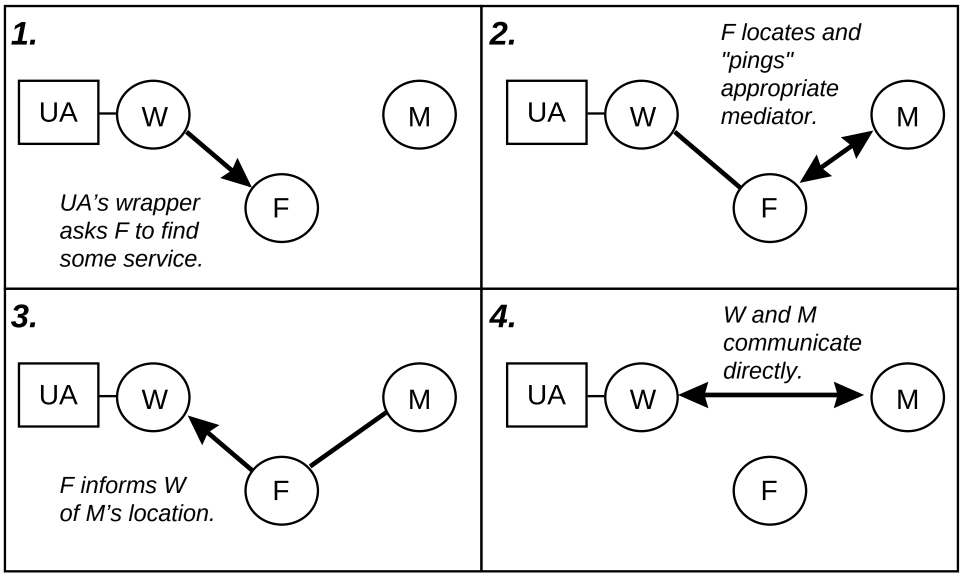

Figure 3. How a facilitator initiates contact between a wrapper and a mediator.

that our hopes are fulfilled, we need to have a precise statement of what the conceptualisation of a given resource is. Recently such an explicit specification of a domain conceptualisation has become known as an "ontology" — largely through the work of Gruber (for example, [11]). In specifying the conceptualisation, the ontology will determine the vocabulary to be used by the resource to which it belongs. Given an ontology it is possible to make an ontological commitment, which Gruber explains as an agreement "to use a vocabulary (i.e. ask queries and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by an ontology" [12]. Provided two resources make the same ontological commitments they will be speaking the same language, and so be in a position to share knowledge. Provided they have an explicit ontology, the harmony of their commitments can be determined.

Note that it is not necessary for the resources to have identical ontologies: all that is required is that they make the same commitments with respect to the knowledge being shared. Thus two resources may both have the concept of *number*, but one may refine this concept into *primes* and *composite numbers*, and the other into *even numbers* and *odd numbers*. The resources both commit to the concept of *number* and can exchange knowledge about *numbers*, even though the more particular knowledge about kinds of numbers that they may have is incommunicable.

Once we recognise that different ontologies may nonetheless make the same ontological commitments, we open up the possibility of translation from the language of one resource to another. In the simplest case a lexical substitution may be all that is needed. For example if one resource has the concept of *even-numbers* and another *numbers-exactly-divisible-by-two*, we can simply substitute the appropriate term to make statements about this concept mutually intelligible. In other cases the translation might not be bidirectional: if one resource has the concept of *even-numbers* and another *powers-of-two*, statements about *even-numbers* will be true of *powers-of-two*, but not necessarily vice versa. The process of translation requires the identification of mismatches between ontologies, and the provision of functions to convert statements made using one vocabulary to statements using the other. For a full discussion of types of mismatch and their implications for translation see [20].

Now the aim of KRAFT is not simply to allow two resources to share knowledge, but to provide for an extensible federation of resources, each potentially with its own conceptualisation, and hence its own ontology. The following strategies are possible:

1. We could stipulate a single KRAFT ontology which would be adopted by all the resources that wished to use the KRAFT network. While this would make things very easy by obviating any need for translation, it is clearly too inflexible. Resources may well need to express things peculiar to themselves, and make ontological commitments which are neither necessary nor desirable for the other resources. Moreover, the task of converting an existing resource to use only the common ontology would make the entry cost to the KRAFT network prohibitive.

2. At the other extreme there would be no shared ontology: each resource would have its own ontology and be capable of translation into the ontologies of the other resources with which it shared knowledge. While

this maximises potential knowledge sharing, it would seem unmanageable for any substantial number of resources. Moreover, the cost of bringing on a new resource, and needing to provide a translation to every other resource would be prohibitive.

3. The third option is for there to be a single KRAFT ontology. Unlike (1), however, each resource would use its own ontology, tailored to its specific needs, but with a translation to the shared ontology provided. Such a shared ontology would represent a kind of lowest common denominator. This seems to be the strategy envisaged by Gruber, and to underlie his criterion that ontologies should make as few commitments as possible [12]. The drawback here is that the shared ontology will be the weakest theory, and hence will restrict knowledge sharing to knowledge which can be expressed in that weakest theory. Moreover, creating such an ontology, able to accommodate a wide, extensible, range of ontologies, would be a difficult task.

4. The fourth option is for there to be small number of shared ontologies recognised by the KRAFT network. If we picture these ontologies as nodes in a directed graph with translations as the edges, the graph must be connected, and there must exist a path from each node to every other node. Here each resource will use its own ontology, and provide a translation to at least one shared ontology. Resources may now communicate through the most appropriate shared ontology, allowing ontologies with many commitments in common to share richer knowledge than those which must rely on the weakest ontology.

It is the fourth strategy that we will be exploring in the KRAFT project, since it seems to provide the best compromise between keeping the number of translations required to a manageable level without being over restrictive as to the knowledge that can be shared.

Within the KRAFT architecture, domain ontology translation is performed by a dedicated mediator (the *ontology mediator*). If other mediators require an ontology translation, they pass the expression to be translated, together with the source and target ontologies, to the ontology mediator which will return the translated expression.

The above all relates to an ontology specifying the conceptualisation of *domain* knowledge. In addition KRAFT will require some other types of ontology. In particular, if facilitators are to be able to select appropriate resources, the capabilities of the various resources must be communicated to them. This will require some common understanding of the various capabilities present in the resources: we feel that this will be best described through a *task* ontology.

6. Current Status and Future Plans

We have implemented an early prototype of the system, distributed among the project sites and incorporating rudimentary mediators and facilitators. The prototype is written in Prolog, using a socket interface for communication [13]. Prolog term structures provide an easily extensible syntax for the current message formats, and Prolog is well suited to doing much of the knowledge transformation work itself.

The initial version of the CCQL protocol is based on a subset of KQML. The choice of a CIF language for interchanging constraints, based on predicates and functions over an object data model, is working well. It allows us to use known mathematical equivalences which guarantee soundness in constraint transformation, while giving a great deal of generality for modelling different kinds of constraint information.

We have brought a database perspective to the original KSE proposal by our use of an object data model to express the knowledge. Thus, we have concentrated on knowledge that can be related to stored data. Although this excludes abstract knowledge of a more literary or philosophical kind, it does allow us to build on the success of entity-relationship models, which are widely used in business data processing systems. The object data model also ties in easily with frame-based knowledge representations used in many expert systems, but excludes the direct modelling of procedural knowledge and triggered rules, as used in these systems. Instead, constraints express the abstract knowledge which would otherwise be hard-coded into procedures and rules. Having the knowledge in the form of constraints facilitates its transformation.

There are interesting issues in adapting query optimisation techniques for distributed queries to make use of constraints; we have encountered some of these in our prototype and they will be the subject of a future paper, as will the work on constraint solving.

The most immediate issues concerning the architecture are the design of the wrappers and the representation of ontologies. The domain of the current KRAFT prototype concerns information about the course regulations and degree requirements of the three universities involved in KRAFT. The idea is to deal with problems such as the eligibility of a student to transfer from one university to another, based on course records to date. This exercise has revealed how often we make implicit assumptions which need to be codified in an ontology. It has also led to a classification of various types of mismatch which can occur between one ontology and another. Finally it has encouraged the adoption of a number of shared ontologies as alternative targets for knowledge fusion (Section 5).

We now need to see how to represent and use this ontological knowledge so that constraint transformations can be carried out by wrappers and mediators. Our current (prototype) domain has demonstrated that often information about a domain is not explicitly recorded and is subject to user interpretation. We are now investigating alternative domains, such as that of the design example given earlier, to study the implications of our approach in a range of different scenarios.

KRAFT is an ongoing collaborative project between research teams at three UK universities (Aberdeen, Cardiff and Liverpool) and a leading international telecommunications company (BT). This partnership of academia and industry is generating a strong focus on strategic issues of knowledge base interoperability which we expect will lead to a wide variety of significant results.

References

[1] W. Behrendt, M. Ashwell, N. Fiddian, and W. Gray. Migration tools for heterogeneous databases in wide-area networks. In *Proceedings of 6th Workshop on Information Technologies and Systems (WITS'96)*, pages 21–30, 1996.

[2] U. M. Borghoff, R. Pareschi, H. Karch, M. Nöhmeier, and J. H. Schlichter. Constraint-based information gathering for a network publication system. In *Proc. 1st Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96) London, U.K.*, pages 45–59, April 1996.

[3] N. Consortium. Reference architecture for the intelligent integration of information. Technical report, NIIIP Consortium, November 1995.

[4] S. Deen. The ANSI/SPARC Architecture and its Implementation in PRECI. In P. Stocker, P. Gray, and M. Atkinson, editors, *Databases — Role and Structure*, pages 93–104. Cambridge University Press, 1984.

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

[6] G. Fahl, T. Risch, and M. Sköld. AMOS — an Architecture for Active Mediators. In *Proc. Workshop on Next Generation Information Technologies and Systems (NGITS'93)*, Haifa, Israel, June 1993.

[7] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proceedings of Third International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press, 1994.

[8] M. R. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[9] P. Gray. Knowledge Re-Use through Networks of Large KBS. In D. Bowers, editor, *Directions in Databases: Proc. 12th British National Conference on Databases, BNCOD'12*, pages 13–22. Springer-Verlag, 1994.

[10] P. Gray. Large Databases and Knowledge Re-use. In I. Wand and R. Milner, editors, *Computing Tomorrow*, pages 110–126. Cambridge University Press, 1996.

[11] T. Gruber. The role of a common ontology in achieving sharable, reusable knowledge bases. In J. A. Allen, R. Fikes, and E. Shandwell, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, Cambridge, MA., Morgan Kaufmann, 1991.

[12] T. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928, 1995.

[13] K. Hui. A Socket Library for Network Communication in Prolog. Technical Report AUCS/TR9703, University of Aberdeen, Department of Computing Science, Mar. 1997.

[14] J. King. *Query Optimisation by Semantic Reasoning*. UMI Research Press, 1984.

[15] Y. Labrou. *Semantics for an Agent Communication Language*. PhD Thesis, Baltimore, Maryland, 1996.

[16] L. Liu, C. Pu, and Y. Lee. An Adaptive Approach to Query Mediation across Heterogeneous Information Sources. In *Proceedings of 1st IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 144–156, Brussels, Belgium, June 1996. IEEE Computer Society Press.

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

[18] N. Singh, M. Genesereth, and M. Syed. A distributed and anonymous knowledge sharing approach to software interoperation. *International Journal of Cooperative Information Systems*, 4(4):339–367, 1995.

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

[20] P. R. S. Visser, D. M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. Analysis of ontology mismatches. In *AAAI Spring Symposium on Ontological Engineering*, 1997.

[21] G. Wiederhold. Intelligent Integration of Diverse Information. In T. Finin, C. Nichola, and Y. Yesha, editors, *First International Conference on Information and Knowledge Management*, Baltimore, USA, Nov. 1992.

[22] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, Mar. 1992.

[23] G. Wiederhold. Mediation and Software Maintenance. Technical Report STAN-CS-TN-95-26, University of Stanford, 1995.

[24] G. Wiederhold. Value-Added Mediation in Large Scale Information Systems. In R. Meersman and L. Mark, editors, *Proceedings of 6th IFIP TC-2 Working Conference on Data Semantics*, Atlanta, USA, May 1995. Chapman and Hall.

[25] G. Wiederhold and M. Genesereth. The Basis for Mediation. In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proceedings of 3rd International Conference on Cooperative Information Systems (CoopIS-95)*, Vienna, Austria, May 1995.