The KRAFT Architecture for Knowledge Fusion and Transformation

Alun Preece, Kit Hui
Department of Computing Science, University of Aberdeen
Aberdeen, UK

Alex Gray, Philippe Marti
Department of Computer Science, Cardiff University
Cardiff, UK

Trevor Bench-Capon, Dean Jones
Department of Computer Science, University of Liverpool
Liverpool, UK

Zhan Cui
Intelligent Business Systems Research, Advanced Communications Research
BT Labs, UK

Abstract

This paper describes the KRAFT architecture which supports the fusion of knowledge from multiple, distributed, heterogeneous sources. The architecture uses constraints as a common knowledge interchange format, expressed against a common ontology. Knowledge held in local sources can be tranformed into the common constraint language, and fused with knowledge from other sources. The fused knowledge is then used to solve some problem or deliver some information to a user. Problem-solving in KRAFT typically exploits pre-existing constraint solvers. KRAFT uses an open and flexible agent architecture in which knowledge sources, knowledge fusing entities, and users are all represented by independent KRAFT agents, communicating using a messaging protocol. Facilitator agents perform matchmaking and brokerage services between the various kinds of agent. KRAFT is being applied to an example application in the domain of network data services design.

1 Introduction and Motivation

Most modern organisations have a number of on-line knowledge sources, distributed among the nodes of their information systems networks. To exploit these knowledge sources to the fullest extent, organisations need to be able to combine knowledge from disparate sources in a highly dynamic way: we call this *knowledge fusion*. There has been an enormous amount of work in distributed information management on the topic of distributed database querying. An important feature of these systems is that they use some kind of *integration schema* to provide a common representation across the entire distributed system. Individual data sources must map between their local schemas and the integration schema in order to combine information in a common form.

A distributed database query retrieves *data instances*, but this is not enough for knowledge fusion. To combine information in a meaningful way, data instances need associated knowledge of their context: how they should be interpreted and how they can be used. This kind of knowledge can often be thought of as "small print", and is typically expressible in the form of *constraints*. As an example, consider a distributed design system for Personal Computers. One on-line resource may be a database of information on available PC operating systems. With a distributed database query, we could discover that one of the available OSes is called "Windows NT"; however, to use this OS instance in a PC design, we would also need to know the "small print" constraint: "Windows NT requires a minimum of 32MB of RAM".

In a knowledge fusion system, the information instances and associated constraints need to be mapped to a common representation to allow meaningful fusion. This requires a *common language* for representing instances and constraints, and a common set of definitions of the *terminology* of the knowledge domain: a *shared ontology* [12].

Consider the following small print constraint, which says that "all PC operating systems called 'winNT' need at least 32MB of RAM":

```
constrain each o in pc_os such that name(o)="winNT"
  to have memory_requirement(o) >= 32
```

This constraint is expressed in CoLan [2], which can serve as a relatively readable, common constraint language. The terminology used (for example, the concept `pc_os`, meaning all PC operating systems, the attributes of this concept, `name` and `memory_requirement`) must be defined in the shared ontology for the PC design domain. Note that, to allow data instances to be represented, the shared ontology must have schema-level information in addition to conceptual-level definitions. For example, the ontology defines the concept `pc_os` as a sub-concept of `pc_software`, which is in turn a sub-concept of `software`. It also defines `pc_os` as having a `name` attribute, the value of which is represented as a string data structure. This is schema-level information which permits instances of `pc_os` to be represented, stored, manipulated, and transmitted across the network.

Constraints such as the example above will not generally be stored in this form within the individual resource. They could appear as database integrity constraints, rules in a rule-based system, slot value restrictors in a frame-based system, or constraints in some other constraint language. Similarly, the terminology used within the individual resource will generally not conform to the shared ontology. For these reasons, knowledge in individual resources needs to be *transformed* to the common language and shared ontology, before it can be fused.

1.1 Operations on a Knowledge Fusion System

There are two main kinds of operation users would wish to perform on a distributed knowledge fusion system:

- *Knowledge Retrieval:* here, users often simply want to find out everything the organisation knows about something. The knowledge fusion system extracts all relevant instances and associated constraints from all available resources, and delivers these in a common form to the user.

  In the PC design domain, for example, the user might ask for all known PC OSes, including any "small print" constraints.

- *Problem-Solving:* here, users not only want the system to fuse knowledge, but also to use the fused knowledge to solve some problem. The system once again extracts the relevant instances and constraints, but now uses these as the basis of a dynamically-constructed constraint satisfaction problem.

  In the PC design domain, for example, the user might specify that they want a PC for real-time video editing; the system will need to extract candidate solution components with attached "small print" constraints, and solve these to construct a suitable PC configuration.

1.2 Services of a Knowledge Fusion System

From the above discussion, it is clear that a knowledge fusion system needs the following services:

- *Knowledge location services:* there must be mechanisms by which a user, and other components, can locate relevant knowledge on the network.

- *Knowledge transformation services:* there must be mechanisms to transform knowledge in individual resources into a common representation language and ontology.

- *Knowledge fusion services:* there must be mechanisms to combine knowledge, and process it — conjoining, simplifying, and finding solutions that satisfy constraints.

These services are the basic requirements of an architecture for building knowledge fusion systems. By implementing software components that provide these services, developers can:

- bring new knowledge sources on-line so that they announce their presence to the network, advertise their services, and commit to providing information in a common representation language and ontology;

- create new knowledge-processing facilities to filter, fuse, adapt, and otherwise add value to the knowledge in the on-line sources;

- construct querying interfaces for users that give them ready access to the rich services of the knowledge fusion system.

The KRAFT project (Knowledge Reuse And Fusion/Transformation) aims to provide such an architecture. The following section presents an overview of the KRAFT architecture; subsequent sections examine the operations of individual components of the architecture.

2 Overview of the KRAFT Architecture

The KRAFT system has an agent-based architecture, in which all knowledge processing components are realised as software agents. An agent-based architecture was chosen for KRAFT for the following reasons:

- Agent architectures are designed to allow software processes to communicate knowledge across networks, in high-level communication protocols; as constraints are a sub-type of knowledge, this was seen as an important feature for KRAFT.

- Agent architectures are highly dynamic and open, allowing agents to locate other agents at run-time, discover the capabilities of other agents, and form co-operative alliances; as KRAFT is concerned with the fusion of knowledge from available on-line sources, these features were seen as being of great value.

The design of KRAFT is consistent with several emerging agent standards, notably the de facto KQML standard [11] and the de jure FIPA standard [6]. Agents are peers; any agent can communicate with any other agent with which it is acquainted. Agents become acquainted by registering their identity, network location, and an advertisement of their knowledge-processing capabilities with a specific type of agent called a *facilitator* (essentially an intelligent yellow pages service).

When an agent needs to request a service from another agent, it asks a facilitator to recommend an agent that appears to provide that service. The facilitator attempts to match the requested service to the advertised knowledge-processing capabilities of agents with which it is acquainted. If a match is found, the facilitator can inform the service-requesting agent of the identity, network location, and advertised knowledge-processing capabilities of the service provider. The service-requesting agent and service-providing agent can now communicate directly.

It is worth emphasising that, while this model is superficially similar to that used in distributed object architectures such as CORBA and DCOM, the important difference is the semantic level at which interactions take place: In distributed object architectures, objects advertise their presence by registering method signatures with registry services, and communicate by remote method invocations. In agent-based systems, advertisements of capabilities are much richer, being expressed in a declarative knowledge representation language, and communication uses a high-level conversational protocol built from primitive conversational actions such as *ask*, *tell*, *advertise*, and *recommend*. Distributed object architectures are in fact highly suitable for implementing agent-based architectures (for example, the ADEPT system used CORBA [9]) but the converse is not true.

A conceptual view of the KRAFT architecture is shown in Figure 1. KRAFT agents are shown as ovals. There are three kinds of these: wrappers, mediators, and facilitators. All of these are in some way knowledge-processing entities.

Wrappers are agents that act as proxies for external knowledge sources, typically databases and knowledge-based systems. These are often legacy systems, so one task of a wrapper is to provide a bridge between the legacy system interface and the KRAFT agent interface. For example, the legacy interface of a relational database will

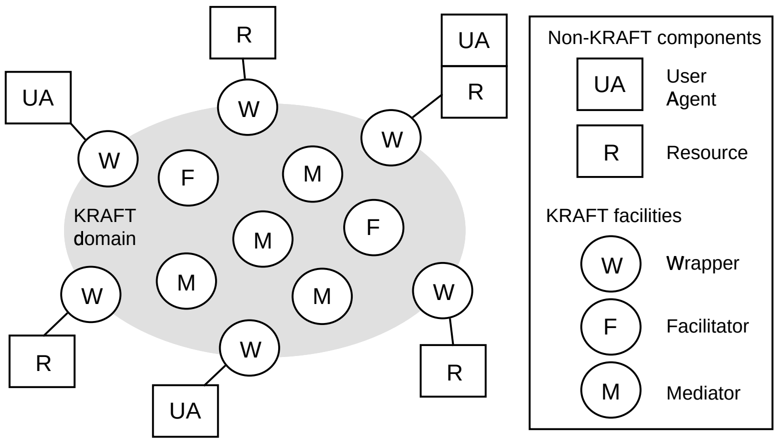


Figure 1: A conceptual view of the KRAFT architecture.

typically be SQL/ODBC; the KRAFT wrapper will accept incoming request messages from other agents in the KRAFT agent communication language, transform these into SQL queries, run them on the database, and transform the returned results to an outgoing message in the KRAFT agent communication language.

Wrappers also provide entry-points into the KRAFT system for *user agents*. User agents allow end-users access to a KRAFT knowledge processing system. A user agent will offer some kind of user interface, with which the user will present queries to the KRAFT network. The user agent will transform the users' queries into the internal knowledge representation language of the KRAFT system, and interact with other KRAFT agents to answer the queries. A user agent will typically also do some local processing on knowledge, at least to transform it for presentation.

Mediators are the internal knowledge-processing agents of the KRAFT system: every mediator adds value in some way to knowledge obtained from other agents. Typical mediator tasks include filtering, sorting, and fusing knowledge obtained from other agents.

Facilitators have already been mentioned above: these are the "matchmaker" agents that allow agents to become acquainted and thereby communicate. Facilitators are fully-fledged knowledge- processing entities: establishing that a service request "matches" a service advertisement requires reasoning with the declarative representations of request and advertisement.

KRAFT agents communicate via messages using a nested protocol stack. KRAFT messages are implemented as character strings transported by a suitable underlying protocol (for example, CORBA IIOP or TCP via sockets). A simple message protocol encapsulates each message with low-level header information including a timestamp and network information.

The body of the message consists of two nested protocols: the outer protocol is the agent communication language CCQL (*Constraint Command and Query Language*) which is a subset of the Knowledge Query and Manipulation Language (KQML) [11].

Nested within the CCQL message is its content, expressed in the CIF protocol (*Constraint Interchange Format*) — a superset of the CoLan constraint language shown in Section 1.

Syntactically, KRAFT messages are implemented as Prolog term structures. An example message is shown below. The outermost `kraft_msg` structure contains a `context` clause (low-level header information) and a `ccql` clause. The message is from an agent called `os_vendor` to an agent called `pc_designer`. The `ccql` structure contains, within its `content` field, an encoded CIF expression (here, we see a "pretty-printed" CoLan constraint; actually, CIF expressions are transmitted in a compiled internal format).

```
kraft_msg(
    context(1,id(19), pc_designer, os_vendor,
        time_stamp(date(21,5,1999), time(15,35,10))),
    ccql(tell, [
        sender : os_vendor,
        receiver : pc_designer,
        reply_with : id(41),
        ontology : shared,
        language : cif,
        content : [
            constrain each o in pc_os
                such that name(o)="winNT"
                to have memory_requirement(o) >= 32
    ])
)
```

Use of Prolog term structures is chiefly for convenience, as most of the knowledge-processing components are written in Prolog. However, the Prolog term structures are easily parsed by non-Prolog KRAFT components; currently there are several components implemented in Java, for example.

As explained in Section 1, the terms used in the CIF part of the message are defined in the shared ontology. To support the representation, storage, and transmission of data instances, the ontology has schema-level information in addition to conceptual-level definitions.

The remainder of this paper is organised as follows: Section 3 explains how the KRAFT facilitator agents work; Section 4 decribes the role of wrapper agents and the common ontology; Section 5 descibes how mediators perform knowledge fusion and associated problem-solving operations; Section 6 discusses an example KRAFT system in a telecommunications design application domain; Section 7 concludes the paper.

3 Facilitator Agents

As introduced above, facilitators provide various routing services within a KRAFT network. As there are many different definitions of facilitators in the literature [1, 3, 4, 10, 14, 16], and the range of services provided by these facilitators varies considerably, this section looks in more detail at exactly how facilitators are used in KRAFT.

In the KRAFT project, the facilitator provides routing functionalities to agents' queries. These basic functions are:

Advertisement handling When the facilitator receives an advertisement from a resource, it is able to process and store this information for further use. An advertisement is a (set of) capabilities that a resource commits to provide. These capabilities are defined in a formalism common to the facilitator and the resources (see Section 3.2).

Facilitation For a given query, this is the action of finding a (set of) resources that comply with a satisfiability (or suitability) criterion. In other words, it is the action of finding a resource whose advertised capability matches the requirements derived from the query. This satisfiability criterion depends on the search strategy adopted for the resolution of the query but it is always somewhere in the spectrum bounded by "the most exact match" on one hand and by "the set of all approximate matches" on the other hand. In other words, the satisfiability criterion is a tradeoff between correctness and completeness in the set of solutions.

3.1 CCQL Facilitation Performatives

As stated in Section 2, CCQL is a subset of KQML [11]. The two main functionalities described in the previous section are supported by the following CCQL operations:

Advertisement Every resource willing to advertise its capabilities does so first by registering with a facilitator (by sending a `register` message containing the references and location of the resource), then by sending an `advertise` message containing the formal description of its capabilities (see Section 3.2). No replies are issued to the originators of these messages unless an error has occured.

Facilitation The protocol encapsulating the facilitation mechanism has two variants:

Forwarding: on receiving a query enclosed in a `recommend_one` (or `recommend_all`) message, the facilitator will reply to the agent at the origin of the query, a singleton (or a list) of matching advertisements (including the references to the corresponding resource). The replied advertisement(s) is contained in a `forward` message.

Brokerage: on receiving a query enclosed in a `broker_one` (or `broker_all`) message, the facilitator will send the enclosed query to the most (or all) relevant resource. The answer to the query is then brought back to the facilitator, which in turn forwards the reply to the originator of the query.

3.2 Expressing capabilities

A resource capability must be represented intentionally for compactness, but in a way that minimises imprecision. The abstract characteristics of a resource are:

- network information to locate and query the wrapper of the resource;
- the CCQL performatives that the wrapper can handle;
- the intentional content of the resource (or a subset of it);
- an abstract representation of the functionality of the resource.

Advertisements are the basic data structure used to communicate these characteristics. The terms used in the body of advertisments are defined in the shared ontology. The possible components of an advertisement are:

- list of allowed CCQL *performatives*;
- list of available *services*, where each service is defined in the shared ontology;
- list of *domains* that the database can deal with, where each domain is defined in the shared ontology;
- specification of a subset of the CIF language, delimiting the expressiveness of the resource query language within CIF as a whole.

The facilitator encapsulates a database of received advertisements with the above components; the CCQL facilitation operations (forwarding and brokerage) are implemented as queries on this advertisement database.

An example CCQL advertisement message follows (the KRAFT message header is not shown). This advertisement is from the wrapper of a PC software vendor called `os_vendor`, sent to a facilitator called `yellow_pages`. The content says that the advertiser can handle CCQL `ask_one` and `ask_all` messages, expressed in a subset of CIF corresponding to SQL queries (from the service description, defined in the shared ontology), about ontology concepts `pc_os` (PC operating systems) and `pc_util` (PC utility software).

```
ccql(advertise, [
    sender : os_vendor,
    receiver : yellow_pages,
    reply_with : id(66),
    ontology : advertise_ontology,
    language : cif,
    content : [
        advertisement.performativeList =
            [ ask_one, ask_all ],
        advertisement.serviceList = [ <database, sql> ],
        advertisement.domainList = [ pc_os, pc_util ] ]
])
```

4 Wrapper Agents and the Shared Ontology

There are three levels of heterogeneity that inhibit the re-use of information stored in resources that we wish to connect to a KRAFT network:

1. *interaction:* different knowledge sources can be interacted with in different ways, e g. some systems only allow the user to pose queries whereas other systems will ask the user for information;

2. *syntactic:* knowledge sources use different representation formats;

3. *semantic:* variations in terminology across knowledge sources.

As outlined in Section 2, interaction and syntactic heterogeneity are addressed by the use of the CCQL and CIF protocols within the KRAFT network and by providing wrapper agents that translate all messages into and out of these protocols. Here we focus on the third of these levels.

To overcome the problem of semantic heterogeneity, a *shared ontology* is specified, which formally defines the terminology of the problem domain. The content of messages within a KRAFT network must be expressed using terms that are defined in the shared ontology. For each knowledge source, a local ontology is specified. For example, where the knowledge source is a database, the local ontology defines the terms that are used in the database schema. Between a local ontology and the shared ontology, there will be a number of *ontology mismatches*, which are instances of semantic heterogeneity [15]. These include the use of different terms to refer to the same concept (i.e. synonyms) and the use of the same term to refer to different concepts (i.e. homonyms). To overcome these mismatches, for each knowledge source an *ontology mapping* is defined. An ontology mapping is a partial function that specifies mappings between terms and expressions defined in a source ontology to terms and expressions defined in a target ontology. To enable bidirectional translation between a KRAFT network and a knowledge source, two such ontology mappings must be defined. We will now describe the format that we use to specify ontology mappings.

In defining an ontology mapping, we begin by specifying a set of ordered pairs or *ontological correspondences*. An ontological correspondence specifies the term or expression in the target ontology that represents as closely as possible the meaning of the source ontology term or expression. For each term in the source ontology, we try to identify a corresponding term in the target ontology. It may not be possible to directly map all of the source ontology terms to a corresponding target ontology term. For some of the terms in the source ontology that cannot be mapped in this way, it may be possible to include them in the ontology mapping by defining correspondences between compound expressions. This leads us to the following classification of ontological correspondences:

1. *class-to-class:* maps a source ontology class name to a target ontology class name;

2. *attribute-type-to-attribute-type:* maps the set of values of a source ontology attribute to a set of values of a target ontology attribute;

3. *attribute-to-attribute:* maps a source ontology attribute name to a target ontology attribute name;
4. *relation-to-relation:* maps a source ontology relation name to a target ontology relation name;
5. *compound-expression-to-compound-expression:* maps compound source ontology expressions to compound target ontology expressions.

As the local and shared ontologies are not represented in the same format that is used for the CIF, the semantic transformation of CIF expressions by wrappers is not based directly on ontology mappings. The relevant ontology mappings are used as part of the specification of a wrapper rather than directly by the wrapper. Consequently, developers have complete autonomy in the implementation of wrappers. In the current KRAFT prototype described in Section 6, we have implemented the transformation of CIF expressions using rewrite rules [8].

A pair of terms and/or expressions in an ontological correspondence are not necessarily semantically equivalent. However, when a wrapper translates a CIF expression, we need to ensure that the target CIF expression is semantically equivalent to the source CIF expression. If this were not the case, constraints passed to a mediator using terms defined in the shared ontology could express very different knowledge than the original constraints expressed in terms defined in the local ontology. We ensure that the semantics of CIF expressions are maintained by defining *pre-* and *post-conditions* for each ontological correspondence. A wrapper that implements an ontology mapping must ensure that these conditions are satisfied when translating the terms in CIF expressions from the source to the target ontology.

5 Mediator Agents and Constraint Fusion

5.1 Constraint Fusion

An application problem in KRAFT is specified by constraint fragments distributed in different resources. Each constraint is part of a conjunctive statement describing the application problem as a constraint satisfaction problem (CSP).

Consider the PC hardware configuration domain: Problem solving knowledge comes from the user requirements, restrictions attached to hardware components from different vendors, and generic design knowledge governing a workable configuration. The following example constraint from the user agent 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. 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
```

The KRAFT approach to this task employs a constraint-fusing mediator which extracts and combines constraints from distributed sources for problem solving purposes. Constraints as abstract mobile objects are transported and transformed to compose a constraint satisfaction problem (CSP), which is then analysed and solved by a combination of distributed database queries and constraint logic programs. With the help of a facilitator, this approach allows tailoring of an execution plan in a dynamic environment, depending on the capability and availability of active online resources.

Our sample constraints give the following fused constraint, which describes the overall requirement on the variables involved:

```
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
```

The reason for fusing the constraint fragments is to provide the basis for exploring how the CSP can best be divided into sub-problems of distributed database queries and sub-CSPs. When a single piece of constraint is insufficient to solve a CSP effectively, we hope to combine information from multiple constraint fragments to arrive at a more solvable solution. It is from the fusion process that useful information can be inferred and captured for problem solving purposes.

5.2 CSP Solving

The constraint fusion process composes a concrete description of the overall CSP in a declarative form. To solve a composed CSP efficiently, a mediator feeds it into a problem decomposer which extracts selection information from the CSP description to generate distributed database queries, with the remaining constraints forming a smaller sub-CSP. The mediator then sends these database queries in multiple messages to different database wrappers to retrieve candidate data values.

Database query generation constitutes an important phase of pre-processing. It shifts part of the problem solving process into the distributed databases by composing data filters as database queries. This prevents unnecessary transportation of irrelevant data into the KRAFT domain and relieves network traffic in a distributed system. Data filtering by database query generation, however, is not sufficient to resolve all constraints. The amount of selection information which can be represented as database queries depends on the expressiveness of the database query language. The remaining sub-CSP has to be resolved by a more powerful constraint solver in the next stage. The final stage of the problem solving process is to feed data and constraints into a constraint solver so that solutions to the CSP can be obtained. In the current prototype, described in Section 6, we use the finite domain constraint solver in the ECLiPSe constraint logic programming (CLP) system.

To form the initial value domains of variables in a CLP program, candidate data retrieved in the previous stage are compiled into CLP data structures. The sub-CSP which is formed by the problem decomposer is then compiled into CLP program codes to impose constraints on these variables. Finally, the mediator sends the CLP program and data to the constraint solver and waits for the result to be returned.

6 An Example KRAFT Application

The KRAFT architecture is being tested on a realistic application in the domain of telecommunications network data services design; this application has been provided by BT. The network data services design problem considered by KRAFT is in the phase of network configuration from the viewpoint of a customer at a single site, allowing a BT network designer to select to meet the customers' requirements: (1) a suitable Point of Presence (POP) at which to connect to the BT network and (2) suitable Customer Premises Equipment (CPE) with which to service the connection (types of CPE include routers, bridges, and FRADs, though it was decided to focus initially solely on router products).

A conceptual view of the application architecture is shown in Figure 2(a). Note that a more detailed description of the network data services design application appears in [7]. In the current implementation, all KRAFT agents (mediators, facilitators, and wrappers) are implemented in Prolog. The user interfaces (user agent and message monitor) are Java applications. The database resources are managed by independent instances of the P/FDM DBMS[1], each with its own local schema. The constraint solver is ECLiPSe. Inter-agent communication is implemented by asynchronous message passing using the Linda model [5].

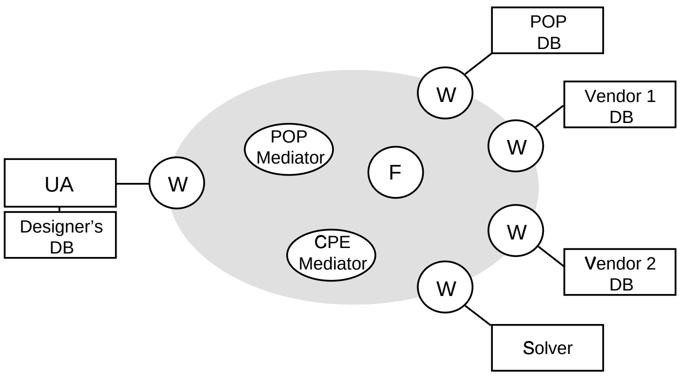
The prototype application employs three sources of knowledge:

- a database of POP information;
- two databases of CPE information: one for each of two competing router vendors.

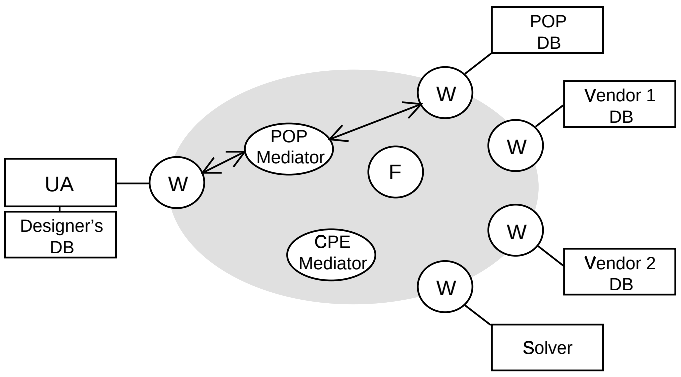
These information sources are considered to be pre-existing legacy databases. For the purposes of the prototype, simplified versions of these databases were created; however, care was taken to ensure that the databases of CPE information were created independently, so as to ensure realistic heterogeneity. Each of the databases was populated with data and constraints; for example, a vendor database was populated with data on the vendor's CPE products, and constraints defining the valid usage of each product. The main aim of creating the three resources was to test the feasibility of creating wrapper agents to transform between the internal knowledge representation (data and constraints) and the KRAFT CIF language.

In the prototype, the tasks of identifying potential POPs and CPEs are the responsibility of mediators. As the two tasks are independent in practice (it is possible

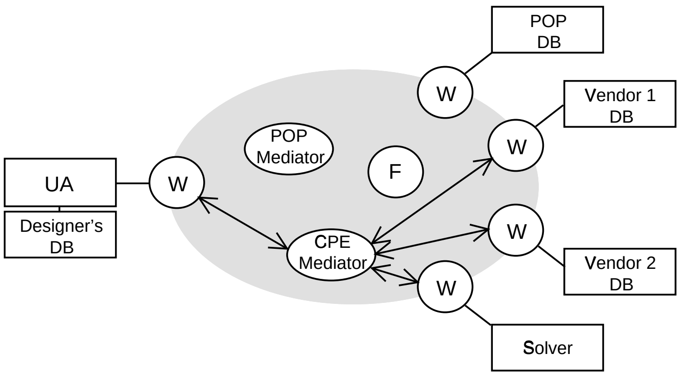
[1] `http://www.csd.abdn.ac.uk/~pfdm`



(a) The Network Services Scenario conceptual architecture



(b) Network Services Scenario interaction 1: locate a POP



(c) Network Services Scenario interaction 2: locate a CPE

Figure 2: A conceptual view of the KRAFT architecture.

to select a CPE on the basis of a customer's LAN and WAN requirements, without knowing which POP will be used, and vice versa), it was decided to provide a separate mediator for each task.

A user agent serves as the front-end to the test system, with a graphical user interface allowing a BT network designer to enter the customer's requirements, and launch two kinds of query into the KRAFT system:

1. For a POP query, the user specifies the location of the customers' site, and the customers' required wide-area network (WAN) services (for example, Frame Relay and ISDN). The user agent formulates the POP query as a KRAFT message, and attempts to locate an agent that can answer the query. It does this by contacting a facilitator, which in turn puts it in contact with the POP Mediator. Upon receipt of the user agent's query, the POP mediator obtains a list of POPs from the POP DB, and filters these according to the user's requirements. It then sends a reply to the user agent. If one or more suitable POPs were found these will be displayed to the user, ranked in order of proximity to the customers' site. These interactions are shown in Figure 2(b).

2. For a CPE query, the user specifies additional constraints on the type of equipment needed, including support for various LAN protocols used within the customer's site (TCP/IP, AppleTalk, 10 base T Ethernet, etc) and support for the required WAN services that determined the choice of POP (Frame Relay, ISDN, etc). Having acquired these constraints, the user agent issues a query to the KRAFT network as above. This time, the CPE Mediator interacts with vendors to select apparently-suitable products, together with any "small print" constraints on these. Fusing these constraints with those from the customer's requirements, the CPE Mediator then calls upon a constraint solver to identify actually-suitable CPE products. These it relays back to the user agent. These interactions are shown in Figure 2(c).

The prototype application has been constructed and is currently under evaluation. Further details on the prototype are available in [7].

7 Conclusion

This paper has described the KRAFT architecture for knowledge fusion and transformation. The generic framework of the architecture is reusable across a wide range of knowledge fusion systems. In addition to the network data services design application described in the previous section, the framework of the KRAFT architecture has been tested in prototype systems for advising students on options to transfer between universities, and advising health care practitioners on drug therapies [13]. Specific software components of the KRAFT system are also reusable, including the CCQL messaging system, facilitators, wrapper shells, and several mediator and solving-related components. Evaluation of the KRAFT architecture is ongoing, and refinements will continue to be made.

Acknowledgements KRAFT is a collaborative research project between the Universities of Aberdeen, Cardiff and Liverpool, and BT. The project is funded by EPSRC and BT. We would like to thank Mike Shave and Jean-Christophe Pazzaglia for their comments on an earlier draft of this paper.

References

[1] Y. Arens, R. Hull, R. King, and M. Siegel, Reference architecture for the intelligent integration of information, Technical report, DARPA — Defense Advanced Research project Agency, August 1995.

[2] N. Bassiliades and P. M. D. Gray, CoLan: A functional constraint language and its implementation, *Data and Knowledge Engineering*, 14:203–249, 1994.

[3] W. P. Birmingham, An agent-based architecture for digital libraries. *D-Lib Magazine*, July 1995.

[4] 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*, pages 45–59, April 1996.

[5] N. Carriero and D. Gelernter, Linda in Context, *Communications of the ACM*, 32:444–458, 1989.

[6] L. Chiariglione, FIPA — agent technologies achieve maturity, *AgentLink Newsletter*, 1:2–4, 1998.

[7] N. J. Fiddian, P. Marti, J-C. Pazzaglia, K. Hui, A. Preece, D. M. Jones, and Z. Cui, Application of KRAFT in data service network design, *BT Technical Journal*, in press.

[8] P. M. D. Gray, S. M. Embury, K. Y. Hui, and G. Kemp, The evolving role of constraints in the functional data model, *Journal of Intelligent Information Systems*, 1–27, 1999.

[9] N. Jennings, P. Faratin, M. Johnson, T. Norman, P. O'Brien, and M. Wiegand, Agent-based business process management, *International Journal of Cooperative Information Systems*, 5:105-130, 1996.

[10] D. R. Kuokka, J. G. McGuire, J. C. Weber, J. M. Tenenbaum, T. R. Gruber, and G. R. Olsen, Shade: Technology for knowledge-based collaborative engineering, *Journal of Concurrent Engineering: Applications and Research*, 1(2), 1993.

[11] Y. Labrou, *Semantics for an Agent Communication Language*, PhD Thesis, University of Maryland, Baltimore MD, USA, 1996.

[12] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout, Enabling technology for knowledge sharing, *AI Magazine*, 12:36–56, 1991.

[13] A. Preece, A. Borrowman, and T. Francis, Reusable components for KB and DB integration, in *ECAI'98 Workshop on Intelligent Information Integration*, 157–168, 1998.

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

[15] P. R. S. Visser, D. M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave, Assessing heterogeneity by classifying ontology mismatches, in *Proc. International Conference on Formal Ontology in Information Systems (FOIS'98)*, IOS Press, 148–162, 1998.

[16] G. Wiederhold, Mediators in the architecture of future information systems, *IEEE Computer*, 38–49, March 1992.