

# Toward a Marketplace Infrastructure for Virtual Organisations

Alun Preece

*University of Aberdeen, Computing Science Department  
Aberdeen AB9 2UE, Scotland  
Phone: +44 1224 272296; FAX: +44 1224 273422  
Email: apreece@csd.abdn.ac.uk*

## Abstract

This paper is a progress report on our work to create an open, flexible infrastructure to support virtual organisations through knowledge fusion technology. Knowledge fusion refers to the process of locating and extracting knowledge from multiple, heterogeneous on-line sources, and transforming it so that the union of the knowledge can be applied in problem-solving. When applied in enterprise-to-enterprise electronic commerce, the knowledge fusion process allows partners to exchange rich business information, and act in an agile and coordinated manner. As an example, we have created a demonstration application in the domain of telecommunications service provision, in collaboration with British Telecom. In our current work, we are extending our infrastructure to include mechanisms to regulate the operational marketplace, to ensure that a virtual organisation complies with the rules of the market.

## Introduction and Motivation

At the AAAI'99 workshop on *Artificial Intelligence for Electronic Commerce* we reported on the KRAFT project (Knowledge Reuse And Fusion/Transformation) and its application to supporting virtual organisations (Preece, Gray, & Hui 1999). KRAFT is intended to be used in an extranet situation where partner organisations exchange business knowledge in a constraint-based format, and form dynamic alliances by finding a mutually-beneficial solution to their various requirement constraints. In KRAFT, constraints are expressed against an object data model, and exchanged using a standard agent communication language (Preece *et al.* 1999). The core KRAFT architecture has the facilities for:

- locating appropriate on-line sources of knowledge;
- transforming heterogeneous knowledge to a homogeneous constraint interchange format;
- fusing the constraints with associated data to form a dynamically-composed constraint satisfaction problem (CSP);
- harnessing existing constraint solver engines to compute CSP solutions.

Recent work in the study of virtual organisations has lead to a commonly-accepted life-cycle for these organisations:

1. *needs identification*: definition of the services or products provided by the virtual organisation;
2. *partner selection*: composition of the group of partners that, together, can meet the identified needs;
3. *operation*: conduct of the transactions by which the services or products are provided by the partners;
4. *dissolution*: disbanding of the group of partners, including any final settlement of payment or other closing transactions.

The KRAFT architecture supports this life-cycle as follows:

1. *needs identification*: customers' requirements can be expressed readily and naturally as a set of constraints; likewise, the capabilities of service and product providers can be expressed using constraints;
2. *partner selection*: by combining and checking the constraints from customers and service/product providers, a virtual organisation can be composed that has the potential to meet the customers' requirements;
3. *operation*: additional constraints will appear during the process of working to satisfy a specific customer's requirements — these may come from the customer itself, or from any of the suppliers; the constraint-solving process can easily accommodate these constraints, dynamically;
4. *dissolution*: the constraint solving process will yield a set of results that include the conditions that must be met when the virtual organisation is disbanded.

The remainder of this paper is organised as follows: the next section reviews the KRAFT architecture; then, we present a walkthrough of a demonstration KRAFT application in the domain of telecommunications service provision (developed with British Telecom); finally, we discuss ongoing work to add facilities necessary to operate regulated electronic markets within KRAFT.

## Review of the KRAFT Architecture

The KRAFT architecture was conceived to support *configuration design applications* involving multiple component vendors with heterogeneous knowledge and data models. This kind of application turns out to be very general, covering not only the obvious manufacturing-type applications (for example, configuration of personal computers or telecommunications network equipment) but also service-type applications such as travel planning (for example, composing package holidays or business trips involving flights, ground travel connections, and hotels) and knowledge management (for example, selecting and combining business rules from multiple heterogeneous knowledge and databases on a corporate intranet).

### Constraints in KRAFT

The most common modern approach to configuration design problems is to tackle them as constraint satisfaction problems (Freuder & Faltings 1999). Where components in the design will come from a number of different vendors, the domains of many of the variables in the CSP are entities stored in each vendor's local product database catalogue. Many of the constraints in the CSP will be on these entity types, defining how the components can be used in configured designs. Some constraints will refer to related instances of other entity types, whose values must be extracted from some other vendor's database and checked for compatibility. Incompatibilities often arise due to the presence of subtle assumptions in vendor's product catalogues — in traditional printed catalogues, these assumptions often appear as “small print”; hence, we refer to this kind of knowledge in KRAFT as *small print constraints*.

As an example, the product catalogue for (fictitious) disk drive vendor, Storage Inc, may have the following small print associated with each of its range of Zip disk drives: *this Zip disk drive requires a PC with a USB-type port*. This kind of small print can readily be expressed as a constraint which can be exchanged with other vendors and resellers:

```
constrain
  each d in disk_drive
    such that name(vendor(d)) =
      "Storage Inc"
    and type(d) = "Zip"
  at least 1 p in ports(host_pc(d))
  to have type(p) = "USB";
```

Similarly to other knowledge-interchange systems (Neches *et al.* 1991), KRAFT requires participating agents to transform their local knowledge to a well-known format, and use a shared ontology of terms; the above constraint is expressed in the KRAFT Constraint Interchange Format (CIF), in the terms of a shared ontology for PC system configuration. These transformations are implemented within a *wrapper* agent for each individual vendor, as part of the setting-up needed for the vendor to join the KRAFT network. Once transformed, the small

print constraints can be fused together with other constraints from various sources, as shown in Figure 1.

In a typical configuration design application, some constraints will be provided by the customer; others will come from the vendors as discussed above; there will also be constraints coming from the service-provider who will act as the configurator of the product or service provided by the application. Typically, the configurator service-provider will be a value-adding reseller from the point-of-view of the component vendors. Note that there may be multiple configurators, each providing a different product or service; also, the design process may have additional stages, where one reseller sells to another reseller, each adding their own constraints to the final product or service. Details of how the constraint fusion process operates within the KRAFT architecture are given in (Preece *et al.* 1999).

### KRAFT Agents

The KRAFT architecture is agent-based:

- *facilitator* agents support the description and location of on-line sources;
- sources are *wrapped* by agent software to transform local knowledge to and from the interchange format;
- *mediator* agents support the querying of sources, and fusion of knowledge from the sources;
- legacy solver engines are provided with agent *wrappers* as front-ends to their services.

An overview of the generic KRAFT architecture is shown in Figure 2. 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. External services are shown as boxes. There are three kinds of these: user agents, resources (typically databases or knowledge bases), and solvers. All of these external services are producers and consumers of knowledge: users supply their requirements to the network in the form of constraints via a user agent service, and receive results in the same way. Resources store, and can be queried for, knowledge and data. Solvers accept CSPs and return the results of the solving process.

KRAFT agents communicate via messages using a nested protocol suite. KRAFT messages are implemented as character strings transported by a suitable carrier protocol: in the current implementation, the carrier protocol is TCP via the socket interface; preliminary work has also been done on an implementation using CORBA IIOP (Preece, Borrowman, & Francis 1998). 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

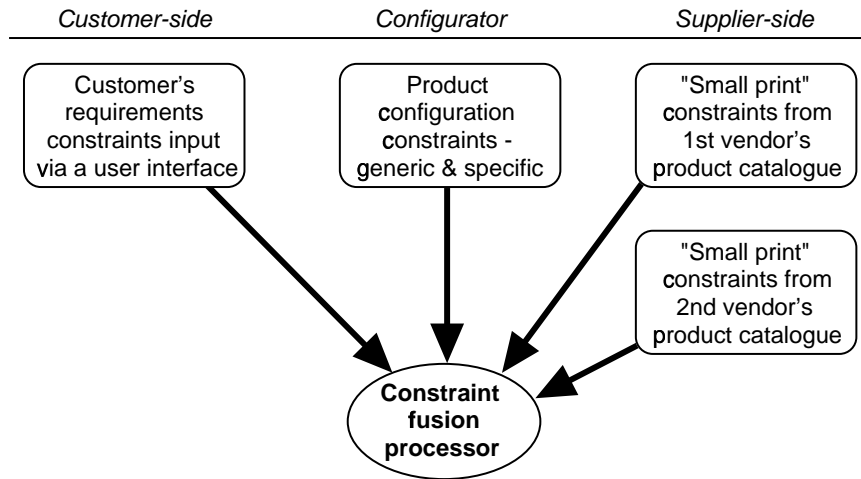


Figure 1: Fusion of constraints from multiple sources.

and Manipulation Language (KQML) (Labrou 1996). Nested within the CCQL message is its content, expressed in the CIF protocol (Constraint Interchange Format).

In the current implementation, the syntax of KRAFT messages is 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 `storage_inc` to an agent called `pc_configurator`. The `ccql` structure contains, within its content field, an encoded CIF expression (here, we see a “pretty-printed” CIF constraint; in the implementation, CIF expressions are actually transmitted in a compiled internal format).

```
kraft_msg(
  context(1,id(19), pc_configurator, storage_inc,
    time_stamp(date(29,9,1999), time(14,45,34))),
  ccql(tell, [
    sender : storage_inc,
    receiver : pc_configurator,
    reply_with : id(18),
    ontology : shared,
    language : cif,
    content : [
      constrain
        each d in disk_drive
          such that name(vendor(d)) =
"Storage Inc"
          and type(d) = "Zip"
          at least 1 p in ports(host_pc(d))
          to have type(p) = "USB"
    ]
  ])
)
```

Use of Prolog term structures is chiefly for convenience, as most of the current knowledge-processing components in the KRAFT implementation 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. It is likely that the next version of the KRAFT implementation will use XML instead of Prolog term structures, as XML retains the ease of parsing, while being a more open interchange standard.

### KRAFT Walkthrough

This section presents an operational walkthrough of the generic KRAFT network shown in Figure 2. The generic network features a user agent  $UA$ , its wrapper  $W_{UA}$ , a facilitator  $F$ , two sample mediators  $M_i, M_j$ , two sample resources  $R_i, R_j$  and their wrappers  $W_{R_i}, W_{R_j}$ , and a solver  $S$  and its wrapper  $W_S$ . In general, of course, there may be multiple user agents, solvers, and any number of mediators and wrapped resources. There may also be multiple facilitators.

The walk-through traces the steps involved in solving a single request, issued by a user to the user agent,  $UA$ . Each numbered step is from the point-of-view of a particular component, named at the start of the step. Messages between components are shown in the form:  $CCQL\text{-performative}(Message\ content) \rightarrow Receiver$

1.  $UA$  submits a request  $Q_{UA}$  in a format local to the user agent.  $Q_{UA}$  will typically be some kind of query, and may include constraints (expressed in the local constraint language).
2.  $W_{UA}$  transforms  $Q_{UA}$  into a KRAFT request  $Q_K$ , in CIF expressed against the shared ontology. Again,  $Q_K$  may include constraints (now expressed in CIF).
3. If  $W_{UA}$  already holds an advertisement  $advertise(A, C_A)$ , where:

$A$  is a named KRAFT agent  
 $C_A$  is a capability of  $A$   
 $C_A$  matches  $Q_K$

Then goto step 5.

Else send message to facilitator  $F$ :

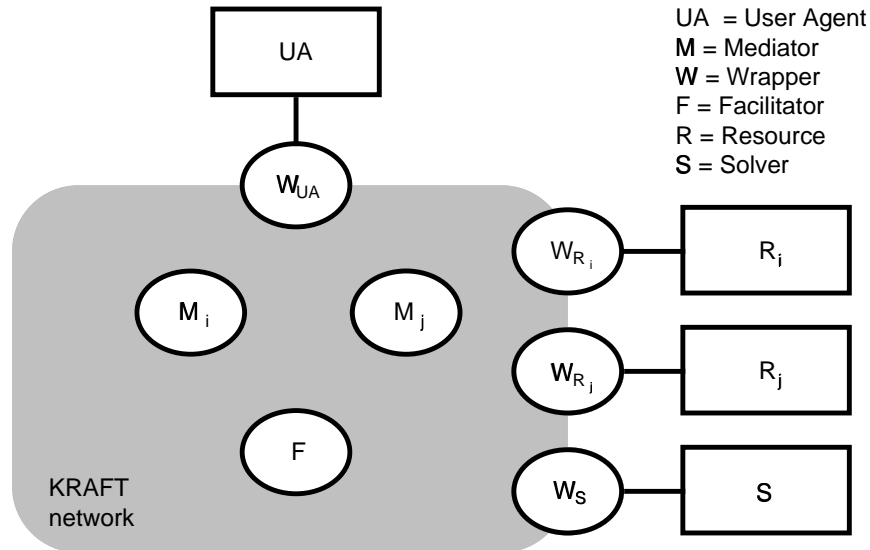


Figure 2: Overview of the generic KRAFT architecture.

$\text{recommend\_}*(Q_K) \rightarrow F$

4.  $F$  searches its directory for an advertisement  $\text{advertise}(A, C_A)$ , where  $C_A$  matches  $Q_K$ , and sends:<sup>1</sup>

$\text{forward}(\text{advertise}(A, C_A)) \rightarrow W_{UA}$

5.  $W_{UA}$  sends  $Q_K$  to the agent identified in the advertisement:

$\text{ask}(Q_K) \rightarrow A$

6.  $A$  processes  $Q_K$  according to the kind of agent it is:

- If  $A$  is a *wrapped resource*,  $W_{R_i}$ :  
 $W_{R_i}$  transforms  $Q_K$  into a local query  $Q_{R_i}$ , in the local ontology, which it submits to the Resource  $R_i$ ; when  $W_{R_i}$  receives the response data  $D_{R_i}$ , it transforms  $D_{R_i}$  to a KRAFT result data object  $D_K$ , in CIF/shared ontology:  
 $\text{tell}(D_K) \rightarrow W_{UA}$
- If  $A$  is a *mediator*,  $M_i$ :  
 $M_i$  decomposes  $Q_K$  into subtasks  $Q_{K_1} \dots Q_{K_n}$ ; then, in parallel, serially, or in some combination thereof,  $A$  recursively performs steps 3–6 with:

each  $Q_{K_i}$  substituting for  $Q_K$   
 $M_i$  substituting for  $W_{UA}$

$M_i$  receives responses, and fuses them into a unified KRAFT result data object,  $D_K$ , in CIF/shared ontology:

$\text{tell}(D_K) \rightarrow W_{UA}$

<sup>1</sup>This walk-through assumes that the agents are using “recommend-style” facilitation (Labrou 1996). KRAFT facilitators also support “broker-style” facilitation, where the facilitator will relay the request  $Q_K$  directly to the advertising agent  $A$  on  $W_{UA}$ ’s behalf.

- If  $A$  is a *wrapped solver*,  $W_S$ :  
 $W_S$  transforms  $Q_K$  into statements in the solver’s local language, which it submits to  $S$ .  
 (\*) If the Solver’s response is a request for more data,  $D_S$ , then  $W_S$ :

transforms  $D_S$  to a KRAFT request,  $Q_{K_S}$   
 recursively performs steps 3–6 with:

$Q_{K_S}$  substituting for  $Q_K$   
 $W_S$  substituting for  $W_{UA}$

receives response(s), transforms them,  
 submits them to the solver,  
 and goes to (\*).

Else  $W_S$ :

transforms response to a KRAFT  
 result data object,  $D_K$ ,  
 in CIF/shared ontology:  
 $\text{tell}(D_K) \rightarrow W_{UA}$

If  $W_S$  needs to recursively perform steps 3–6 as noted above, then in performing step 3 it is possible that the solver’s wrapper will consult the facilitator to find an agent that can handle its data request; however, it is likely that the solver’s wrapper will direct its requests for more data back to the originator of  $Q_K$  (probably a mediator). This is because the mediator will likely be constructing variable domains on behalf of the solver.

7.  $W_{UA}$  receives the KRAFT result object, transforms it into the local format, and passes it to  $UA$  for display.

## Related Work

The design of the KRAFT architecture builds upon recent work in agent-based distributed information systems. In particular, the roles identified for KRAFT

agents are similar to those in the InfoSleuth system (Bayardo 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 the Xerox work focusses upon the use of constraints to support querying of distributed data sources, rather than the extraction of constraints from distributed sources, and the use of these constraints in configuration design problem-solving.

The Smart Clients project (Torrens & Faltings 1999) is related to KRAFT in the way they conduct problem-solving on a CSP dynamically specified by the customer, using data extracted from remote databases. Their approach differs from KRAFT in that only data is extracted from the remote databases, no small print constraints come attached to the data; also, all the problem-solving is done on the client, rather than by mediator agents. No constraints are therefore transmitted across the network; conversely, it is the constraint solver that is transmitted to the client's computer, to work with the constraints specified locally by the customer.

Finally, ongoing work at IBM's T. J. Watson Research Center is similar in concept to KRAFT's use of small print constraints (Reeves *et al.* 1999). The difference is that this work uses a rule-based formalism to specify contractual "fine print" in the form of business rules. Logic programming techniques are then used to reason with the rules.

## KRAFT Virtual Organisation Demonstration

The KRAFT architecture has been instantiated with a realistic application in the domain of telecommunications network data services design; this application was specified by the KRAFT project's industrial partner, 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 services and equipment to meet the customers' requirements:

- A suitable Point of Presence (POP) at which to connect to the BT network.

- 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.

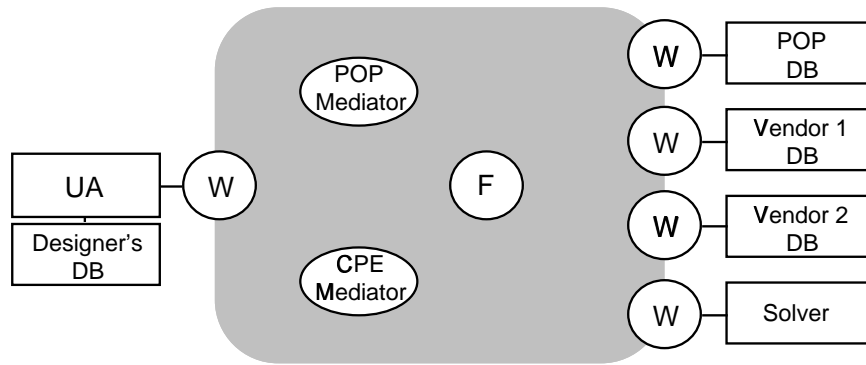
A conceptual view of the application architecture is shown in Figure 3(a). This application maps onto the generic architecture shown in Figure 2 as follows:

- A single wrapped *User Agent*, designed by BT, provides a user interface for the two kinds of request listed above. Coupled to this user agent is a database of designer knowledge, which will be accessed during the network data services configuration design process. The *User Agent Wrapper* provides network access to and from both the user agent and the *Designer's DB*.
- As the two kinds of request are independent (it is possible 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: the POP request is handled by the *POP Mediator*, and the CPE request is handled by the *CPE Mediator*.
- There is a single *Facilitator* which is not specific to the application domain, except that it has access to the shared ontology.
- There are four wrapped resources:
  - *POP Database*, a database of POPs (based on BT's own POP database);
  - *Vendor 1 DB*, a product catalogue database for a CPE vendor (based on the actual product catalogue of 3Com);
  - *Vendor 2 DB*, a product catalogue database for a second CPE vendor (based on the actual product catalogue of Cisco).
  - *Designer's DB*, a source of network data services design constraints (based on knowledge acquired from BT network data services designers).
- There is a single wrapped legacy constraint *Solver* engine.

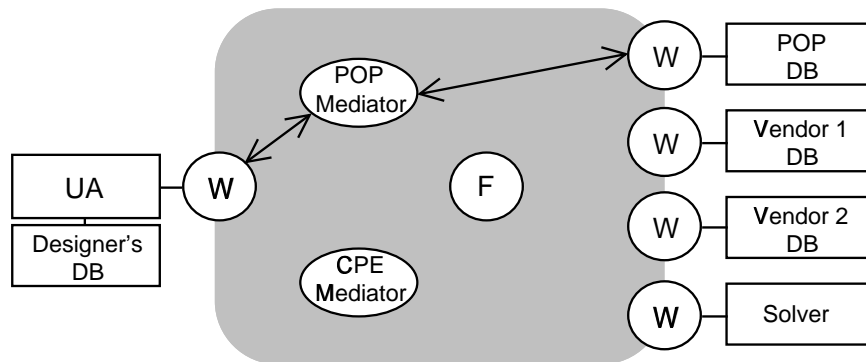
All the agents (mediators, facilitators, and wrappers) are implemented in Prolog. The user interfaces (user agent and a message monitor) are Java applications. The database resources are managed by independent instances of the P/FDM DBMS<sup>2</sup>, each with its own local schema. The constraint solver is ECLIPSe. Inter-agent communication is implemented by asynchronous message passing using the Linda model (Carriero & Gelernter 1989).

The four wrapped resources 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

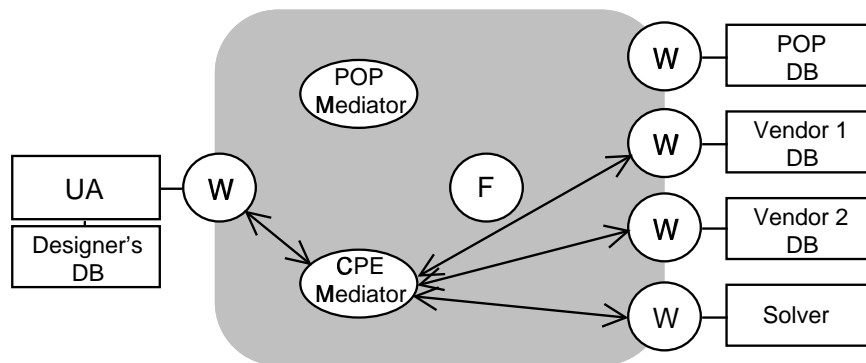
<sup>2</sup><http://www.csd.abdn.ac.uk/~pfdm>



(a) KRAFT Network DataServices application architecture



(b) KRAFT Network DataServices application interaction 1: locate a POP



(c) KRAFT Network DataServices application interaction 2: choose CPE

NOTE In (b) and (c), interactions with the facilitator are not shown.

Figure 3: A conceptual view of the KRAFT demonstration application.

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 four resources was to demonstrate the feasibility of creating wrapper agents to transform between the internal knowledge representation (data and constraints) and the KRAFT CIF language.

When the various service-providing agents come on-line, each sends an appropriate `advertise` message to the facilitator:

- *POP DB Wrapper* advertises that it can supply *POP* data objects;
- *Vendor 1 Wrapper* advertises that it can supply *router* data objects where the manufacturer is "Vendor 1";
- *Vendor 2 Wrapper* advertises that it can supply *router* data objects where the manufacturer is "Vendor 2";
- *Solver Wrapper* advertises that it can process finite domain CSPs;
- *POP Mediator* advertises that it can supply information on POPs that are closest to a given location;
- *CPE Mediator* advertises that it can supply CPE data objects from multiple vendors that meet given customer requirements.
- *User Agent Wrapper* advertises that it can supply network data services design constraints.

### Handling POP Requests

A POP request issued by the user agent results in the following sequence of actions, summarised in Figure 3(b):

1. Via the *User Agent*, the user specifies the location of the customer's site, and the customer's required wide-area network (WAN) services (for example, Frame Relay and ISDN).
2. The *User Agent Wrapper* formulates the POP query as a KRAFT message, and attempts to locate an agent that can answer the query by contacting the *Facilitator* through a `recommend CCQL` message, indicating that it needs to find a POP closest to a given location.
3. The *Facilitator* matches the *User Agent Wrapper*'s request to the advertisement from the *POP Mediator*, and forwards the matching advertisement back to the *User Agent Wrapper*.
4. The *User Agent Wrapper* sends an `ask-one` message to the *POP Mediator*, requesting a POP that meets the user's requirement constraints (location and services).

5. The *POP Mediator* contacts the *Facilitator* to find a source of POP data, and is forwarded the advertisement from the *POP DB Wrapper*. It then sends an `ask-all` message to the *POP DB Wrapper*, requesting all POP data objects with the required services.
6. Assuming that the *POP DB Wrapper*'s reply was non-empty, the *POP Mediator* computes which POPs are nearest the customer's site, and sends these data objects in a `tell` message to the *User Agent Wrapper*. This computation is simple enough that the *POP Mediator* performs it itself, and does not need to invoke the *Solver*.
7. Upon receipt of the data from the *POP Mediator*, the *User Agent Wrapper* transforms it to the local format for presentation to the user via the *User Agent* itself.

### Handling CPE Requests

A CPE request issued by the user agent results in the following sequence of actions, summarised in Figure 3(c):

1. Via the *User Agent*, 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).
2. The *User Agent Wrapper* interacts with the *Facilitator* as above, this time looking for vendor-independent CPE data objects. It is forwarded the *CPE Mediator*'s advertisement.
3. The *CPE Mediator* receives an `ask-all` request from the *User Agent Wrapper*, specifying all the customer's requirement constraints. It sends a `recommend-all` message to the *Facilitator* to discover all CPE vendors currently on-line.
4. The *Facilitator* finds no CPE vendors have advertised but, knowing from the shared ontology that *router is a kind of CPE*, it is able to forward *CPE Mediator* the advertisements from *Vendor 1 Wrapper* and *Vendor 2 Wrapper*.
5. The *CPE Mediator* uses some of the customer's requirement constraints to formulate `ask-all` requests to each vendor's wrapper. Each wrapper responds, telling the *CPE Mediator* the router data objects that meet the given requirements, and any attached "small print" constraints on these router data objects.
6. The *CPE Mediator* formulates a CSP by fusing the constraints it now has:
  - all the customer requirement constraints;
  - all the "small print" constraints on router data objects from both vendors;

- network data services design constraints which it obtains by sending an `ask-all` message to the *User Agent Wrapper*, having discovered its location from the *Facilitator*.
7. The CSP is formulated as a finite domains CSP, so the *CPE Mediator* interacts with the *Facilitator* to discover a finite domain solver. It then sends the *Solver Wrapper* the CSP.
  8. Assuming there is at least one solution to the CSP, the *Solver Wrapper* tells the solution set to the *CPE Mediator*, which then returns these results to the *User Agent Wrapper*.
  9. The user can examine the solutions (if any) via the *User Agent* and, if necessary, refine the constraints and invoke further requests to the KRAFT network.

The application has been constructed and experiments conducted with it. Background details are available in (Fiddian *et al.* 1999). The application shows the feasibility of the KRAFT approach to supporting distributed configuration design systems where vendors are able to advertise their product catalogues to resellers, who in turn offer value-adding services to customers via customised user agents.

## Adding Marketplace Support to KRAFT

The current version of the KRAFT architecture lacks some important mechanisms needed to support virtual organisations fully: there is nothing to ensure that a transaction between partners is properly conducted, and closed. To date, we have assumed that all partners are fully cooperative and trustworthy. This is clearly a naive assumption, so we are now extending the KRAFT architecture with mechanisms to enforce good behaviour among participating agents. At the same time, we are working to make the architecture more open and robust. This work-in-progress is outlined below. A future paper will report on the final results.

### Marketplace Mediation Services

We are introducing a new type of mediator — the *Marketplace Mediator* — which has responsibility for enforcing the rules and policies of the marketplace among all participating agents. Interactions between agents are now separated into two kinds:

- *Information-seeking operations* are essentially just for “browsing” or “window shopping”. There are no guarantees being made in these interactions, and no agent can be held accountable for inaccuracy, incompleteness, or impermanence of information obtained. These are essentially the current type of interaction in KRAFT systems; no marketplace mediator is involved in these operations. Agents typically use these operations to explore provisional solutions,

before deciding whether to initiate a business transaction.

- *Business transactions* are governed by well-defined (but flexible) interaction protocols that result in committed deals between agents. These operations must be conducted through a marketplace mediator. Each participant in a business transaction sacrifices some of its autonomy to the marketplace mediator, which runs the interaction protocol on behalf of the marketplace.

When an agent wishes to initiate a business transaction with one or more other agents, it uses the facilitation services to find a marketplace mediator that will run the desired interaction protocol. We have built three interaction protocols to date:

- *Fixed-price buy/sell*: This is the simplest form of deal, where the seller states a price which the buyer either accepts or rejects. Upon stating its price, the seller will be committed to that price if the buyer accepts it. Upon accepting the price, the buyer is committed to paying. If either party breaks its commitment, the marketplace mediator will record this behaviour in a *reputation database* so that it will have bearing on future deals. (In the extreme case, the offending agent will be banned from future dealing in the marketplace.)
- *Negotiated-price buy/sell*: This protocol is an extension of the above simple case, to allow the buyer and seller to “haggle” over price. Either side may agree to suggest the initial price. Stated prices are binding as before.
- *English-auction buy/sell*: This protocol implements an English auction with multiple buyers and a single seller. Here, the marketplace mediator is performing the role of an auction house and auctioneer.

These interaction protocols will form the links in electronic supply chains. For example, in the telecommunications service provision scenario, we could envisage the following links:

- the customer enters into a negotiated-price buy/sell with the service reseller;
- the service reseller enters into a negotiated-price buy/sell with the POP operator;
- the service reseller enters into a fixed-price buy/sell with the CPE vendor.

We are currently completing the implementation of a generic marketplace mediator which is capable of being instantiated with interaction protocols. This infrastructure is being built in Java over the Jini framework, to allow runtime discovery of marketplace services, and fully dynamic binding/rebinding between agents.

## Conclusion

This paper has described the KRAFT agent-based architecture for supporting virtual organisations. The

generic framework of the architecture is reusable across a wide range of knowledge processing systems, including applications in electronic commerce and knowledge management, where various partners ally themselves together because they wish to interact by exchanging constraints. The prototype network data services application has proven the concept of supporting virtual organisations by constraint fusion. However, the current architecture assumes an unreasonably high degree of trust and cooperation between partners so we are extending it to include robust mechanisms to enforce the rules and policies of an electronic marketplace.

**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.

## References

- Andreoli, J.; Borghoff, U.; and Pareschi, R. 1995. Constraint agents for the information age. *Journal of Universal Computer Science* 1:762–789.
- Bayardo, R. 1997. InfoSleuth: agent-based semantic integration of information in open and dynamic environments. In *Proc. SIGMOD'97*.
- Carriero, N., and Gelernter, D. 1989. Linda in context. *Communications of the ACM* 32:444–458.
- Fiddian, N. J.; Marti, P.; Pazzaglia, J.-C.; Hui, K.; Preece, A.; Jones, D. M.; and Cui, Z. 1999. A knowledge processing system for data service network design. *BT Technology Journal* 14:117–130.
- Freuder, E., and Faltings, B., eds. 1999. *Configuration: Papers from the AAAI-99 Workshop*. Menlo Park, CA: AAAI Press.
- Labrou, Y. 1996. *Semantics for an Agent Communication Language*. Ph.D. Dissertation, University of Maryland, Baltimore MD, USA.
- Neches, R.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.; and Swartout, W. 1991. Enabling technology for knowledge sharing. *AI Magazine* 12(3):36–56.
- Preece, A.; Hui, K.; Gray, W. A.; Marti, P.; Bench-Capon, T.; Jones, D.; and Cui, Z. 1999. The kraft architecture for knowledge fusion and transformation. In *Research and Development in Intelligent Systems XVI (Proc ES99)*, 23–38. Springer.
- Preece, A.; Borrowman, A.; and Francis, T. 1998. Reusable components for KB and DB integration. In *Proc. ECAI'98 Workshop on Intelligent Information Integration*, 157–168. ECCAI.
- Preece, A.; Gray, P.; and Hui, K. 1999. Supporting virtual organisations through knowledge fusion. In *Artificial Intelligence for Electronic Commerce: Papers from the AAAI-99 Workshop*. Menlo Park, CA: AAAI Press.
- Reeves, D.; Grosz, B.; Wellman, M.; and Chan, H. 1999. Toward a declarative language for negotiating executable contracts. In *Artificial Intelligence for Electronic Commerce: Papers from the AAAI-99 Workshop*. Menlo Park, CA: AAAI Press.
- Torrens, M., and Faltings, B. 1999. Smart clients: constraint satisfaction as a paradigm for scaleable intelligent information systems. In *Artificial Intelligence for Electronic Commerce: Papers from the AAAI-99 Workshop*. Menlo Park, CA: AAAI Press.