A knowledge processing system for data service network design

N J Fiddian, P Marti, J-C Pazzaglia, K Hui, A Preece, D M Jones and Z Cui

The knowledge reuse and fusion/transformation system (KRAFT) is research prototype software for combining and transforming constraint-based knowledge. It is being developed in collaboration with BT by three UK universities — Aberdeen, Cardiff and Liverpool. The KRAFT system is designed to help its users to locate relevant data and constraint knowledge held in distributed heterogeneous data and knowledge bases, convert this knowledge to generate a required composite problem specification, and exploit appropriate constraint solvers in solving the specified problem. The system architecture utilises intelligent software agent technology in the form of wrapper, facilitator and mediator agents for co-operative knowledge processing, and also a shared data model and a shared ontology as the basis for knowledge exchange.

This paper describes the application of KRAFT in the design of data service networks for BT. It presents, in turn, an introduction including an overview of the KRAFT system architecture, a description of the BT network design test application scenario, the application of KRAFT to this scenario, conclusions and further work.

1. Introduction

1.1 Motivation and related work

Agent-based architectures are proving to be an effective approach to developing distributed information systems [1], as they support rich knowledge representations, meta-level reasoning about the content of on-line resources, and open environments in which resources join or leave a network dynamically [2]. KRAFT employs such an agent-based architecture [3] to provide the required extensibility and adaptability in a dynamic distributed environment.

Unlike most agent-based distributed information systems, however, KRAFT focuses on the exchange of data and constraints among agents in the environment it supports.

Recent research in the area of software agent technology offers promising ways of supporting distributed design applications, but the area is still far from mature. Early projects such as PACT [4] and SHADE [5] showed that agent technology could support the exchange of rich business information (using the knowledge interchange format (KIF)) between organisations using heterogeneous technologies, with a limited amount of organisational agility — basic 'matchmaking' brokerage connecting suppliers to customers. While demonstrating the promise of the agent-based approach, these projects revealed problems — the complexity of the KIF representation has prevented it from gaining widespread use, while the limited brokerage model hinders the implementation of flexible negotiation schemes.

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 [1]; 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 (KSE) [6], in that some of the KSE facilitation and brokerage methods are employed, along with a subset of the 1997 Knowledge Query and Manipulation Language (KQML) specification [7].

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.

1.2 Overview of the KRAFT system architecture

The KRAFT system has an agent-based architecture, in which all knowledge-processing components are realised as software agents. As the benefits and features of agent-based software architectures have been reported widely elsewhere [8—10], these will not be dwelt upon here. However, it is worth noting why an agent-based architecture was chosen for KRAFT:

- agent architectures are designed to allow software processes to communicate knowledge across networks, in high-level communication protocols — since 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 KQML [11] and FIPA [9, 10]. 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 [12] and DCOM [13], 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 [8]), but the converse is not true.

A conceptual view of the KRAFT architecture is shown in Fig 1. KRAFT agents are shown as shaded ovals or rectangles according to whether they are internal or external to the KRAFT domain, respectively. There are four kinds of these — user agents, wrappers, mediators and facilitators. All of these are in some way knowledge-processing entities. The (external) knowledge resources with which they work are shown as unshaded rectangles in Fig 1.

User agents provide end users with entry-points into a KRAFT knowledge-processing system. A user agent will offer some kind of user interface, through which the user will present queries to the KRAFT network. The user agent wrapper will transform user queries into the internal knowledge representation language of the KRAFT system,

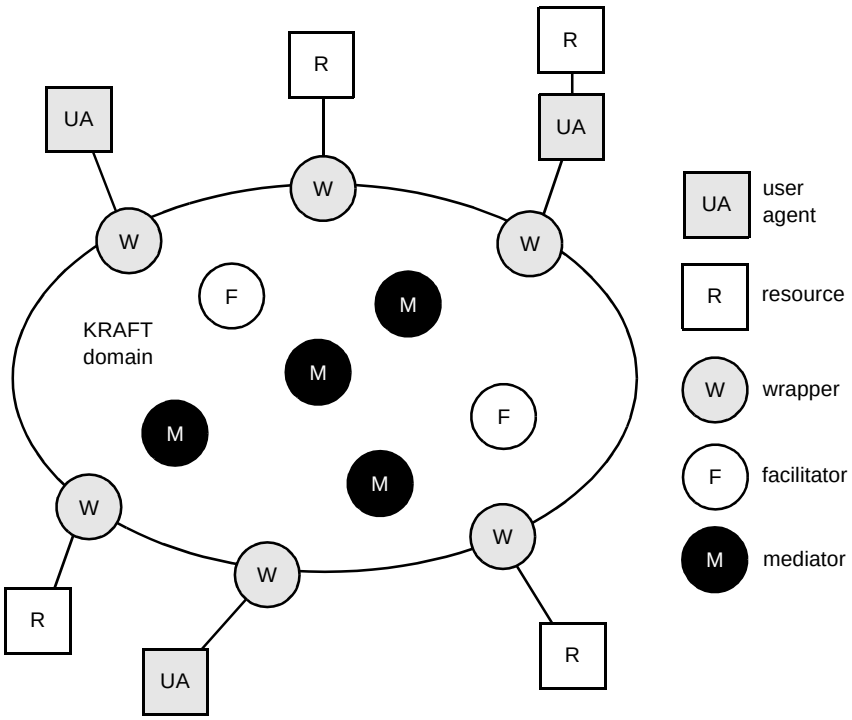


Fig 1 The KRAFT conceptual architecture.

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.

Wrappers are agents that act as proxies for user agents and for external knowledge resources, typically databases and knowledge-based systems. As these resources are often legacy systems, 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 typically be SQL/ODBC. The associated KRAFT wrapper will accept incoming request messages from other agents in the KRAFT agent communication language, transform these into SQL queries, and run them on the database. Finally it transforms the returned results to an outgoing message in the KRAFT agent communication language.

Mediators are the ultimate 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 other agents to become acquainted and thereby communicate. Facilitators are knowledge-processing entities — establishing that a service request 'matches' a service advertisement requires reasoning with the declarative representations of request and advertisement, respectively.

KRAFT agents communicate via messages using a nested protocol hierarchy. 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 one is the agent communication language CCQL (Constraint, Command and Query Language) which is a subset of the 1997 specification of the KQML [7]. Nested within the CCQL message is its content, expressed in the CIF protocol (constraint interchange format). Figure 2 shows the anatomy of a KRAFT message and will be useful in understanding the message sequences presented for illustration purposes in the following sections of this paper.

It is worth noting that, syntactically, KRAFT messages are implemented as Prolog term structures. This 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 of these implemented in Java, for example.

2. The BT network design application scenario

The network design application problem area was defined to be, on the one hand, simple enough to build in a reasonable period of time without the need for a great deal of knowledge acquisition, and, on the other hand, complex enough to test the important features of the KRAFT architecture. It was decided to define the problem from the viewpoint of a customer at a single site, allowing the BT network designer, as a KRAFT user, to select components to meet the customer's requirements:

- a suitable point of presence (POP) for connection to the BT network,

- a suitable customer premises equipment (CPE) product with which to service the connection (types of CPE include routers, bridges and FRADs, though it was decided to focus initially on router products).

A user agent was required 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 queries into the KRAFT system. This interface is shown in Fig 3.

- POP query

  For a POP query, the user specifies the location of the customer's site, and the customer's required WAN services (for example, Frame Relay and ISDN). The

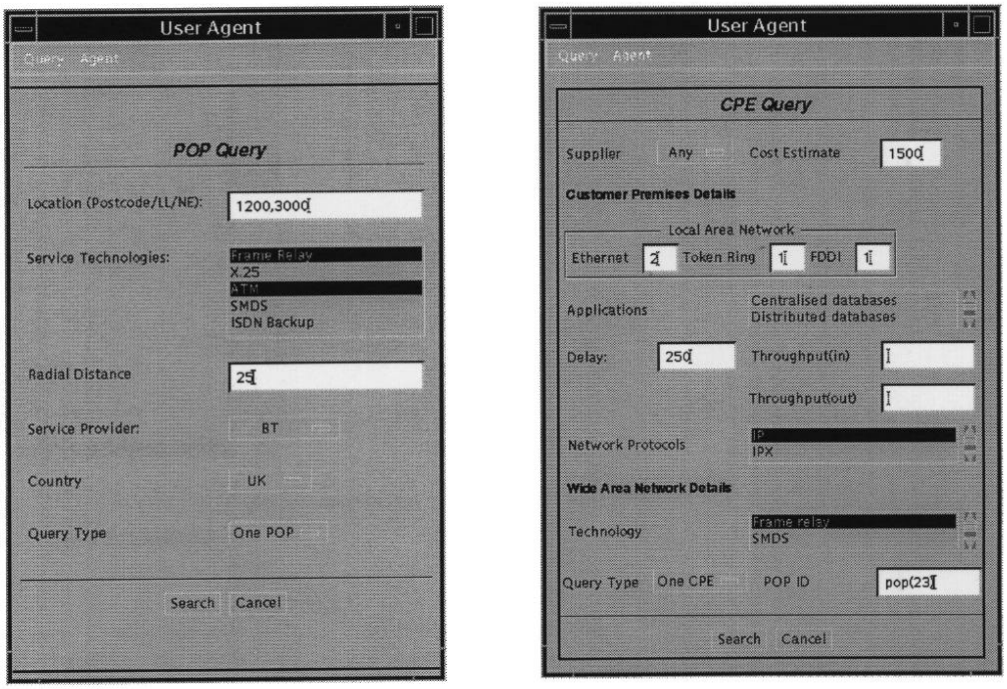
```
kraft_msg(
   context(1,id(19),krl(bt,ua),krl(bt,ua),krl(aberdeen,cpe_m),
     time_stamp(date(15,10,1998),time(16,45,10))),
   ccql(ask_one,[
   sender : krl(bt, ua),
   receiver : krl(aberdeen, cpe_m),
   reply_with : id(61)
   ontology : shared,
   language : cif,
   content : [
     [var(c)]:[string],[generate(route,var(r)),
       restrict(product_code,[router],[var(r)],var(c))]]
])
```

header

CCQL

CIF

Fig 2 Anatomy of a KRAFT message.



(a) (b)

Fig 3 Interfaces of the BT user agent for querying (a) a POP, then (b) a CPE.

user agent, through its KRAFT wrapper, must formulate the POP query as a KRAFT message, and attempt to locate an agent that can answer the query. It will do this by contacting a facilitator, as described in the previous section. Assuming that a suitable service-providing agent is found, the query will be sent to that agent. If the query is answered successfully, a reply message will eventually be received by 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 customer's site.

- CPE query

  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, 10BaseT, 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. However, assuming that a choice of CPE is returned by reply, the user agent has the final task of creating design configurations for the possible solutions (combinations of LAN configuration + CPE + POP), and allowing the user to explore these.

This problem, when suggested, appeared well-suited to solution by a KRAFT system. First of all, it was clear that solutions would need to employ at least three sources of knowledge:

- a database of POP information,
- two databases of CPE information — one for each of two competing vendors.

Moreover, it was entirely reasonable to consider these information resources as legacy databases, as they were known to pre-exist. For the purposes of the test, simplified versions of these resources were created. However, care was taken to ensure that the databases of CPE information were created independently, in order to ensure realistic heterogeneity. Each of these 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) of the resources and the KRAFT CIF language.

Here are some examples of vendor database constraints:

```
constrain each r in isdn_router
   so that at least 1 p in ports(r)
   so that some proto in protocol(p)
   has name(proto)= "ISDN"

constraing each r in serial_router
   so that at least 1 p in ports(r)
   so that some proto in protocol(p)
   has serial_supported(proto)= "serial"
```

In the KRAFT model, the tasks of identifying potential POPs and CPEs are the responsibility of mediators. As the two tasks are independent in practice (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 conceptual architecture of the test system is shown in Fig 4; the main interactions between agents in solving the two test problems (choose POP in Fig 5, choose CPE in Fig 6) are also shown.

3. Application of KRAFT to the BT scenario

3.1 Interaction 1 — choose POP

The first part of the BT network design application — find a suitable POP — using KRAFT consists of two main stages:

- the registration stage,
- the resolution stage, which itself contains three sub-stages:

  — the recommendation stage,

  — the querying stage,

  — the answering stage.

Registration

This is an initialisation stage where the agents are spawned in the KRAFT domain and where each resource/ wrapper and each mediator registers with a facilitator and then advertises its capabilities.

A typical registration message is prefixed by the `register` performative and contains information necessary to identify the resource. Once sent, this information is stored in the facilitator.

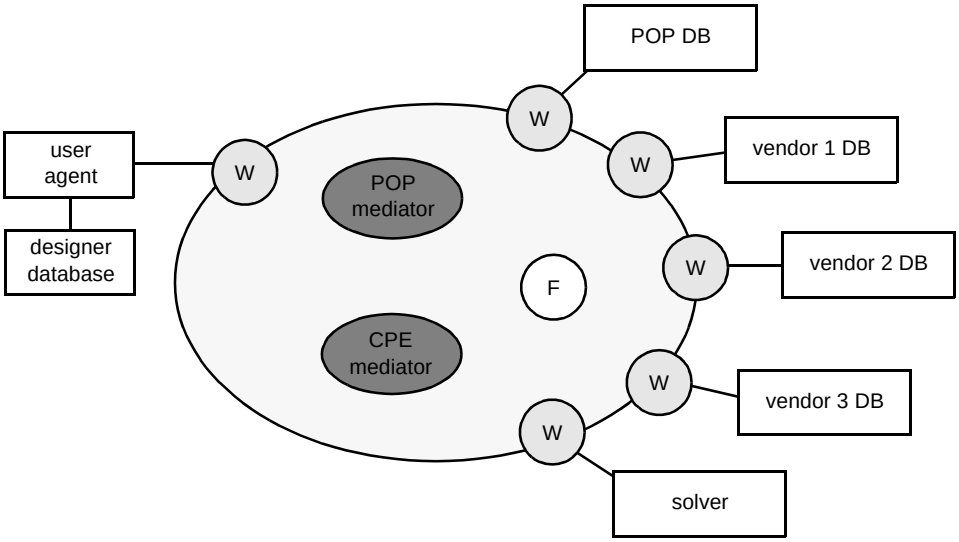


Fig 4 BT network design application architecture.

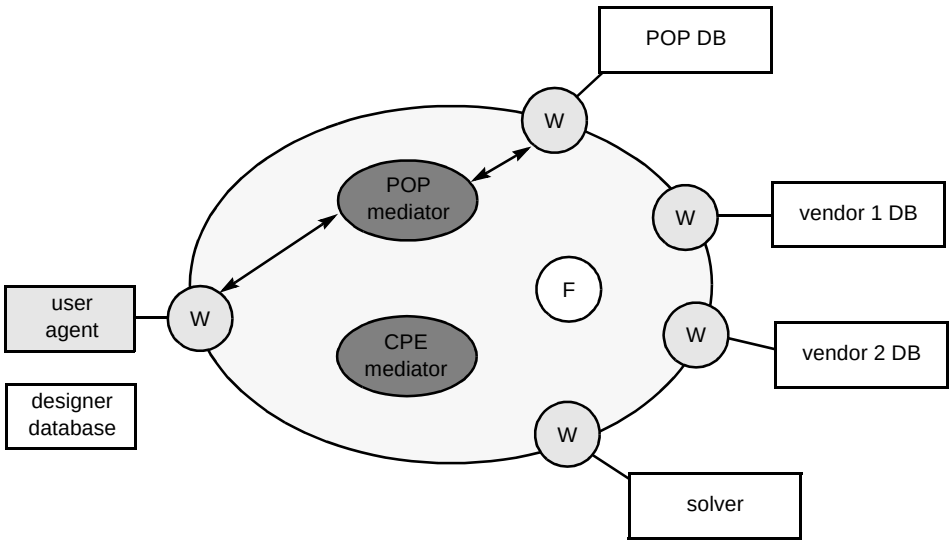


Fig 5 Interaction 1 — select a suitable POP.

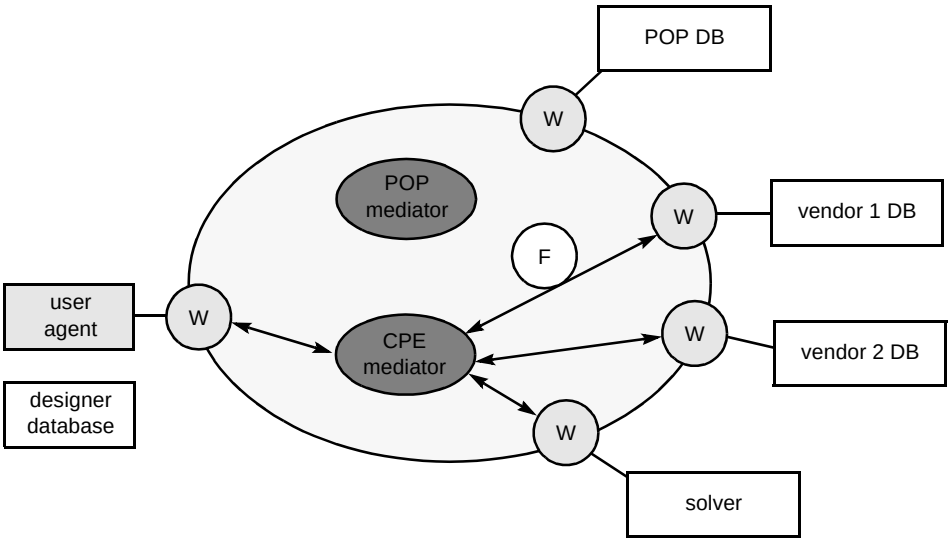


Fig 6 Interaction 2 — select a suitable CPE.

An advertisement message is prefixed by the `advertise` performative and contains an abstract definition of the capabilities of the advertising resource, that is:

- a signature of the functionalities made available,
- or the classes defining the data that the resource can handle or provide.

Here is an example of a resource (the `popdb_wrapper`) advertising its capabilities (the following messages are simplified versions of the actual ones):

```
advertise:
     sender: (liverpool,popdb_wrapper)
     receiver: (cardiff,facilitator)
     content:
            performative = ask_one
             class = pop
```

This message calls for a new entry to be made in the advertisement database of the facilitator and declares that queries prefixed by `ask_one` can be handled by the resource. Moreover, the resource/wrapper agent capabilities are advertised as a set of possible requests that deal with a certain class (the `class` field of the advertisement has been set, whereas the `pattern` field has been left blank). This is known as class-based advertisement. In other circumstances, capabilities could be advertised as a set of possible CIF expressions that a resource/wrapper or a mediator can handle. This is known as pattern-based advertisement. Here is an example, advertising the capabilities of the Liverpool `pop_mediator` :

```
advertise:
     sender: (liverpool,pop_mediator),
     receiver: (cardiff,facilitator),
     content:
            performative = ask_one
            pattern = get_pops(integer,
            integer,set_of(string),integer)
```

This message specifies a CIF expression of the pattern-type `get_pops(...)` as a well-formed query that the `pop_mediator` at Liverpool can handle.

Resolution

As indicated previously, the resolution stage consists of three sub-stages — firstly, recommendation which allows the querying agent to locate the relevant facility (the resource/wrapper or the mediator able to answer its query), then, the actual querying of the newly found facility, and finally, the answering stage where the (set of) solution(s) is retrieved and returned to the original querying agent.

As will be shown, it is possible to nest a resolution stage inside a querying sub-stage just as it is possible to nest goals within a sub-goal in any goal-directed resolution algorithm.

Recommendation

This first sub-stage is when a consumer agent (a mediator or a user agent), in this case the BT user agent, sends a query to the facilitator as a request to the latter to identify a suitable facility capable of meeting its consumer knowledge.

```
recommend_one:
      sender:  (bt,user_agent)
      receiver:(cardiff,facilitator)
      content:
            performative =     ask_one
            class =            pop
            function_pattern = get_pops
```

The facilitator answers with a `forward` message containing the original advertisement that matches the pattern contained in the foregoing `recommend_one` message.

```
forward:
   sender: (cardiff,facilitator)
   receiver: (bt,user_agent)
   content:
     advertise:
       sender: (liverpool,pop_mediator),
       receiver: (cardiff,facilitator),
       content: function_pattern = get_pops
```

Querying

Now that the relevant facility (in this case, a mediator) has been identified, the user agent can query it. The following code is the CCQL part of this query and it asks the `pop_mediator` for a list of POPs (thanks to the `get_pops` function built into the mediator) which are equipped with frame relays and are within a 10 000 units radius of the geographic *x,y* co-ordinates (615760,244340):

```
ask_one:
      sender: (bt,user_agent)
     receiver: (liverpool,pop_mediator)
     content:
           get_pops(615760, 244340,
            ['Frame Relay'],10000) : pop
```

This query will trigger a sub-resolution attempt by the mediator and consequently a new `recommend_one/forward` exchange between the mediator and the facilitator occurs. These two messages will not be examined in detail but they are nevertheless displayed in the screenshot sequence (Fig 7).

This sub-resolution stage will also trigger a subquery (`ask_all`) from the `pop_mediator` to the `popdb_wrapper`. Indeed, the mediator needs a list of the possible POPs to be able to filter out the bad candidates (i.e. those not equipped with frame relays or outside the specified geographical radius). Here is the code of the query to the database of POPs:

```
ask_all:
   sender: (liverpool,pop_mediator)
   receiver: (liverpool,popdb_wrapper)
   content:
      generate all (Pop,Xpop,Ypop) where
            supported_protocol(Pop) = 'Frame
                        Relay',
          location_x(Pop) = Xpop,
          location_y(Pop) = Ypop.
```

Less formally, this code queries the POP database through the `popdb_wrapper` to return every stored instance of the class `pop` (together with their *x,y* co-ordinates) which supports frame relay.

Figure 8 shows the sub-resolution and resolution stages of Interaction 1 (choose POP) unwinding as they return their results.

Answering

Here are the contents of the `tell` message issued by the `popdb_wrapper` (the core of the contents only is shown):

```
[[ var(id)='Aberdeen', var(xloc)=394100,
                       var(yloc)=806420 ],
 [ var(id)='Birmingham', var(xloc)=407720,
                       var(yloc)=287880 ],
 [ var(id)='Bristol', var(xloc)=358900,
                       var(yloc)=173100 ],
  .
  .
  .
 [ var(id)='Ipswich', var(xloc)=615760,
                       var(yloc)=244340 ],
  .
  .
  .
 [ var(id)='Telford', var(xloc)=369540,
                       var(yloc)=309220 ],
 [ var(id)='York', var(xloc)=460400,
                       var(yloc)=452700 ]
]
```

This message is then exploited by the `pop_mediator` which will send the final result back to the BT user agent. The final answer can be seen in the contents part of the following CCQL `tell` message:

```
tell:
     sender: (liverpool,pop_mediator)
     receiver: (bt,user_agent)
     content:
           P : pop
           P = 'Ipswich'
```

This `tell` message completes Interaction 1 (choose POP).

3.2 Interaction 2 — choose CPE

In this second part of the BT network design application of KRAFT, the user composes a CPE query by specifying the POP selected in the first part and other parameters such as the required communication protocol support, bandwidth and transmission delay. The task of this part of the problem-solving process is to find a usable CPE that satisfies all these requirements. This involves:

- problem-solving knowledge,
- customer specifications,
- designer constraints that define a usable configuration of equipment products,
- restrictions that are attached to these products by their vendors as small-print constraints.

The KRAFT approach to this task employs a constraint-fusing mediator which extracts and combines constraints from distributed sources. 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

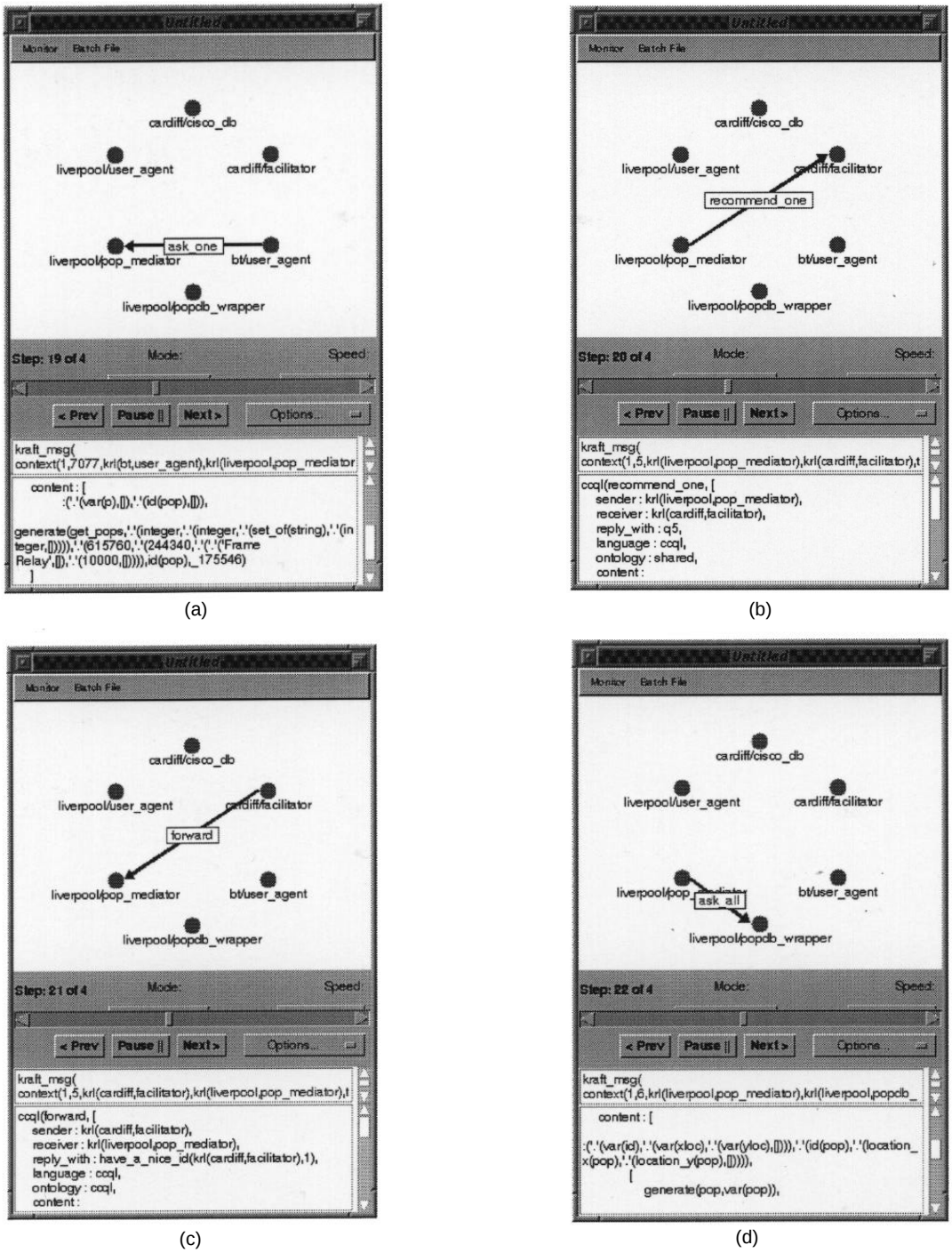


Fig 7 An `ask_one` message (a) to the `pop_mediator` results in an `ask_all` message (d) to the `popdb_wrapper` .

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 on-line resources.

This section presents a higher level description of the interaction between KRAFT agents and the activities within them, in contrast to the detailed treatment of agent intercommunication presented in the previous section. Also highlighted are the major steps that constitute important elements in this approach which make KRAFT a suitable system for solving configuration problems in a distributed environment.

Constraint gathering

The CPE query from the user agent represents one of the many problem-solving constraints that come from different sources and the problem-solving process starts when this query reaches the CPE mediator (CPE-M). Before an execution plan can be decided, the CPE-M needs to acquire a complete description of the problem by gathering all problem-solving knowledge from the following sources.

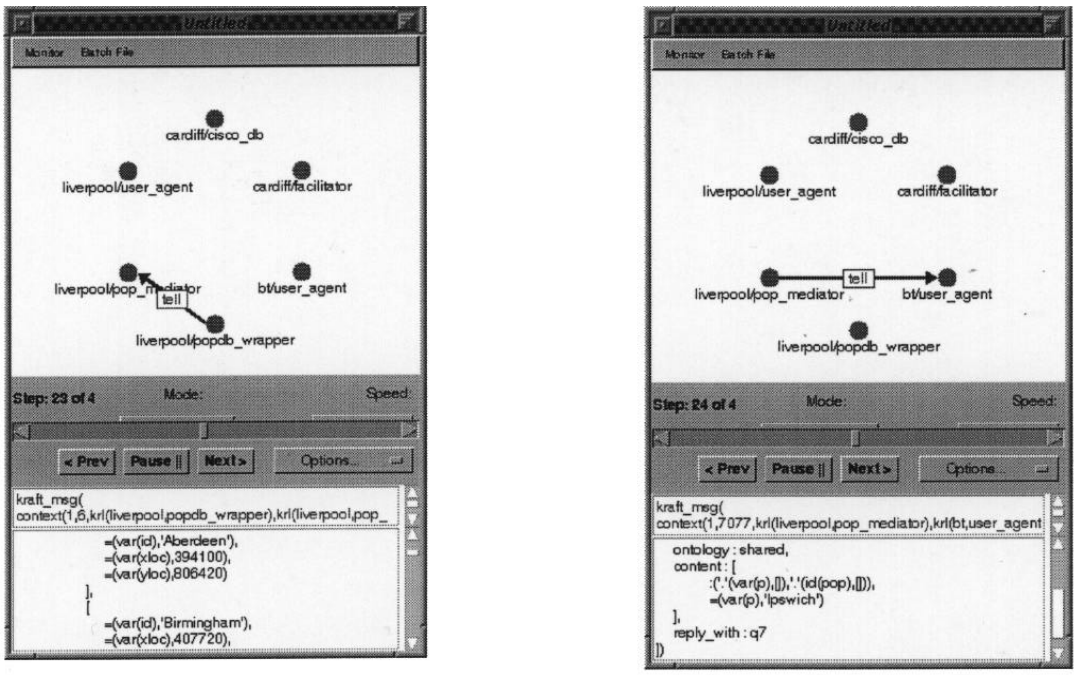


Fig 8 The result of the `ask_all` message is returned to the `pop_mediator` and processed to produce a result which is returned to the BT user agent.

- Constraints from the customer

  These constraints originate as design parameters and are specified by filling in parameter values in the BT user agent GUI (see Fig 3). These values are then used to compose the constraints into a CPE query which is sent to the CPE-M to initiate the whole problem-solving process of Interaction 2. The following is a customer constraint restricting all potential network design solutions to have a required-bandwidth of 123 units:

```
constrain each nd in network_design_n1
   each cr in customer_requirements(nd)
to have value(req_bandwidth(cr))=123
```

- Designer constraints

  These constraints represent generic restrictions on composing a usable configuration relating to a particular application problem. In the current prototype, designer constraints are stored in the user agent as canned queries. The following is an example of a designer constraint that requires all routers in a usable network configuration to have a bandwidth greater than or equal to the required value:

```
constrain each nd in network_design_n1
          each cr in customer_
                      requirements(nd)
          each rb in req_bandwidth(cr)
          each cp in customer_
                          premises(nd)
          each e in equipment(cp)
            such that e is a router_n1
          each rp in port(e as router_n1)
            such that rp is a router_
                              port_n1
          each b in bandwidth(rp as
                        router_port_n1)
to have value(rb) =< value(b)
```

- Small-print constraints from vendors

  Small-print constraints resemble the footnotes in a product catalogue restricting the use of a particular item of equipment. Conceptually, these constraints are attached to their corresponding equipment products and must be satisfied when the equipment is used. They are stored in different vendor equipment databases. The following is a small-print constraint in the 3COM database putting extra requirements on a router when it is in the 'superStack II NETBuilder SI 44x U' series:

```
constrain
   each s in router_n1
    such that series(s)= "superStack II
         NETBuilder SI 44x U" and name
         (manufacturer(s))="3Com"
   each c in connector(port(s)) such
                that c is a ethernet_n2
to have value(line_speed(c)) in {10,100}
    and name(system_of_measure(line_
                        speed(c)))="bps"
    and units(system_of_measure(line_
                        speed(c)))="M"
```

Resource location

In the KRAFT prototype, the CPE query encapsulates customer and designer constraints in a single message. To retrieve the third category of problem-solving knowledge (small-print constraints), the CPE-M consults the facilitator for relevant resources by sending a number of `recommend_all` messages. This allows the CPE-M to locate all vendor equipment databases containing small-print constraints and candidate CPE values.

Constraint extraction

When the resource locations come back as `forward` messages from the facilitator, the CPE-M composes meta-queries to extract constraints from the databases concerned. This is accomplished by sending `ask_all` messages to the corresponding database wrappers.

Constraint transformation

As outlined in section 2, a KRAFT wrapper performs bidirectional translation of messages, using ontology mappings to specify semantic transformation:

- requests to knowledge resources are transformed into a local representation for processing by the knowledge resource,
- the results generated by the knowledge resource are transformed from the local representation into a KRAFT message for transmission to the querying agent.

For example, a request from the `pop_mediator` is transformed by the `popdb_wrapper` into a form that can be processed by the associated `popdb`. The results of this query are then transformed by the `popdb_wrapper` into a KRAFT internal representation and are passed as the value of the result message 'content' field to the `pop_mediator`. The transformation performed by a wrapper addresses two types of heterogeneity:

- syntactic heterogeneity — knowledge resources use different representation formats,
- semantic heterogeneity — variations in terminology occur across knowledge resources.

This section focuses on the second of these. For the purpose of this discussion, it is assumed that when translating an expression from a knowledge resource representation to the KRAFT internal representation, the syntactic transformation is performed before the semantic transformation and vice versa for expressions passed in the opposite direction. Thus all semantic transformation is performed on expressions represented in the KRAFT internal format.

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 the KRAFT domain must be expressed using terms that are defined in the shared ontology. For each knowledge resource, a local ontology is specified. For example, where the knowledge resource 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 [14]. 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 resource an 'ontology mapping' is defined. An ontology mapping is a function that maps from expressions using terms defined in a source ontology to expressions using terms defined in a target ontology. To enable bidirectional translation between a KRAFT domain and a knowledge resource, two such ontology mappings must be defined. The format that is used to specify ontology mappings is now described. In defining an ontology mapping, firstly a set of ordered pairs — 'ontological correspondences' — is specified. 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, an attempt is made 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 to the following classification of ontological correspondences:

- class-to-class — maps a source ontology class name to a target ontology class name,
- 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,
- attribute-to-attribute — maps a source ontology attribute name to a target ontology attribute name,
- relation-to-relation — maps a source ontology relation name to a target ontology relation name,
- 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 form part of the specification of a wrapper rather than part of its implementation. Consequently, developers have complete autonomy in the implementation of wrappers. In KRAFT, the transformation of CIF expressions is being implemented using rewrite rules [15].

A pair of terms and/or expressions in an ontological correspondence are not necessarily semantically equivalent. However, when a wrapper translates a CIF expression, it must be ensured that the target CIF expression is

semantically equivalent to the source CIF expression. If this were not the case, constraints passed to the CPE-M using terms defined in the shared ontology could express very different knowledge about a vendor's products 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 CIF expressions from the source to the target ontology.

Semantic transformation of constraints

Having outlined the method of semantic transformation of expressions, this is now related to the transformation of constraints in the BT network design application. Firstly, the query from the CPE-mediator to the vendor databases that requests constraints must be translated to the local representation for each vendor database. Since this is a meta-data query, there are no semantic mismatches between the shared and local versions of the query. Once the syntactic conversion of the query has been performed, the wrapper passes the query to the resource. The wrapper next needs to translate the results of the query into the shared form. This section focuses on the semantic aspects of this transformation by giving an example of the semantic translation of a constraint.

The following constraint is the local form of one of those that satisfies the query passed to the 3COM wrapper:

```
constrain each s in superStack_II_
                        NETBuilder_SI_44x_U
          each rp in router_inv(s) such
           that port_type(rp)= "Ethernet"
          each i in interface(rp)
          to have line_speed(i) in {10,100}
```

This constraint expresses the information that for the specified set of routers (`SuperStack_II_NETBuilder_SI_44x_U` routers), the transmission speed of the Ethernet port is either 10 Mbit/s or 100 Mbit/s. The translation of this constraint relies on the wrapper implementing an ontology mapping which specifies correspondences in the shared ontology for the local ontology terms. Not all such correspondences will be given here, but the approach will be illustrated with a suitable example.

To enable the translation of the local ontology term `superStack_II_NETBuilder_SI_44x_U` to the shared ontology, the following correspondence is specified:

```
Source  : superStack_II_NETBuilder_SI_44x_U
Target  : router_n1
```

Although the term `router_n1` is semantically the closest term in the shared ontology for the given local ontology term, it is not semantically equivalent to it. Unless the mapping is specified further, it cannot be guaranteed that the shared ontology version of the above constraint will express the same knowledge as the local original. To this end, the following post-condition to be applied by the wrapper is then added:

```
Post : [r:router_n1],[name(manufacturer(r))
                    ="3Com"
       and series(r)= "superStack II
                    NETBuilder SI 44x U"]
```

Now when the wrapper translates the first line of the constraint:

```
constrain each s in superStack_II_
                      NETBuilder_SI_44x_U
```

it will append this additional information to ensure that the translated constraint is semantically equivalent to the local version, giving:

```
constrain each s in router_n1
      such that series(s)= "superStack II
             NETBuilder SI 44x U" and name
             (manufacturer(s))= "3Com"
```

Further, such correspondences and conditions used in the implementation of the wrapper will ensure that the rest of the constraint is also translated to the shared ontology in a semantically equivalent fashion. The whole constraint expressed in terms of the shared ontology is:

```
constrain each s in router_n1 such that
      series(s)= "superStack II NETBuilder
                                SI 44x U"
      and name(manufacturer(s))= "3Com"
   each c in connector(port(s)) such that
                        c is a ethernet_n2
   to have value(line_speed(c)) in {10,100}
      and name(system_of_measure(line_
                          speed(c)))="bps"
      and units(system_of_measure(line_
                            speed(c)))= "M"
```

Constraint fusion

Constraint fragments in KRAFT represent problem-solving knowledge distributed among different sources. Each of them is part of a conjunctive statement that imposes restrictions on the variables involved. When all constraints have arrived, the CPE-M fuses them together to form a complete CSP description which summarises all requirements on the solutions.

The original sample constraints give the following fused constraint:

```
constrain each nd in network_design_n1
    each cp in customer_premises(nd)
    each e in equipment(cp) such that e is
                              a router_n1
    each rp in port(e as router_n1) such
               that rp is a router_port_n1
    each cr in customer_requirements(nd)
to have value(req_bandwidth(cr)) =< value
(bandwidth(rp as router_port_n1))
    and value(req_bandwidth(cr))=123
    and if  series(e)= “superStack II
             NETBuilder SI 44x U” and name
             (manufacturer(e))= “3Com”
        then each c in connector(port(e))
          such that c is a ethernet_n2 has
          value(line_speed(c)) in {10,100}
          and name(system_of_measure(line_
          speed(c)))=“bps” and units
          (system_ of_measure(line_speed
          (c)))=“M”
        else true
```

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

CSP solving

The constraint-fusion process composes a concrete description of the overall CSP in a declarative form. The next step is to solve the composed CSP by retrieving candidate data values from relevant databases and testing them against the required constraints. This solving process has two characteristics:

- variables in the CSP are fed with candidate data values from distributed databases — data retrieval must be involved, although data filtering techniques can be applied to reduce the resultant traffic,
- candidate values are tested against the required constraints for their validity — a reasoning mechanism is needed to eliminate invalid values and find the CSP solutions.

These two characteristics highlight a key concept in KRAFT — the duality of constraints as database query filters and CSPs. It allows constraint information to migrate between the two paradigms and to be solved by the collaboration of constraint solvers and databases.

Distributed database query generation

To solve the composed CSP efficiently, the CPE-M 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 CPE-M then sends these database queries in multiple `ask_all` messages to different vendor equipment database wrappers to retrieve candidate data values.

In our current example, the CPE-M deduces from the fused constraints that no router with a bandwidth less than 123 units is acceptable as a solution. As a result, it generates the following database query to retrieve the manufacturer and product-code of all potential routers that satisfy the bandwidth requirement:

```
for each e in equipment
    each rp in port(e as router_n1)
        such that value(bandwidth(rp as
                router_port_n1)) >= 123
print(manufacturer(e),product_code(e));
```

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.

Constraint logic programming code generation

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. The current prototype uses the finite domain constraint solver in the ECLiPSe constraint logic programming (CLP) system.

A typical CLP system, like ECLiPSe, works by solution elimination. A CLP program starts by declaring the initial domains of variables and removes invalid values by posting constraints on these variables. Variables are finally instantiated to obtain a consistent solution of the CSP. Backtracking through the labelling process will get all solutions to the problem.

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 CPE-M sends the CLP program and data to the constraint solver and waits for the result to be returned:

```
[ [var(r)=router_n1(12)],
      [var(r)=router_n1(34)] ]
```

(The numbers in parentheses are instance identifiers of the selected routers.)

This last `tell` message completes Interaction 2 (Choose CPE).

4. Conclusions and further work

The emphasis of this paper has been on providing a practical description of the application of KRAFT to a BT network design scenario. This scenario was chosen as a suitable test of the KRAFT system architecture because it offered sufficient scope to instantiate and evaluate each component of the architecture as well as providing a good example of constraint fusion, one of the original aspects of the project. A prototype KRAFT system was developed for testing purposes and has been employed in successfully evaluating the KRAFT architecture as this paper has shown.

A number of issues have been raised in the process which require further research. Among the most important of these are issues of:

- system scalability and robustness,
- ontology structuring and evolution,
- application diversity.

The most significant contribution of the KRAFT project has been:

- to highlight the value of constraints as a distinct category of knowledge in their own right,
- to develop techniques for processing such knowledge flexibly and profitably in a distributed environment of multiple heterogeneous knowledge resources.

References

1 Bayardo R et al: 'Infosleuth: agent based semantic integration of information in open and dynamic environments', in Proceedings of Sigmod'97 (1997).

2 Wiederhold G and Genesereth M: 'The basis for mediation', in Proceedings of the 3rd International Conference on Cooperative Information Systems (COOPIS'95) (1995).

3 Gray P M D et al: 'KRAFT: knowledge fusion from distributed databases and knowledge bases', in Proceedings of the 8th International Workshop on Database and Expert Systems Application (DEXA), IEEE Computer Society, pp 682—691 (1997).

4 Cutkosky M, Engelmore R, Fikes R, Genesereth M, Gruber T, Mark W, Tenenbaum J and Weber J: 'PACT: an experiment in integrating concurrent engineering systems', IEEE Computer, 26, No 1, pp 8—27 (1993).

5 Kuokka D, McGuire J, Weber J, Tenenbaum J, Gruber T and Olson G: 'SHADE: technology for knowledge-based collaborative engineering', Journal of Concurrent Engineering: Applications and Research (CERA), 1, No 2, (1993).

6 Neches R, Fikes R, Finin T, Gruber T, Patil R, Senator T and Swartout W: 'Enabling technology for knowledge sharing', AI Magazine, 1, No 12, pp 36—56 (1993).

7 Labrou Y: 'Semantics for an agent communication language', PhD thesis, University of Maryland Graduate School. Baltimore, Maryland (September 1996).

8 O'Brien P D and Wiegand M E: 'Agents of change in business process management', BT Technol J, 14, No 4, pp 133—140 (October 1998).

9 O'Brien P D and Nicol R C: 'FIPA — towards a standard for software agents', BT Technol J, 16, No 3, pp 51—59 (July 1998).

10 FIPA: 'Agent communication language', Technical report. Foundation for intelligent physical agents (1997).

11 Mayfield J, Labrou Y and Finin T: 'Evaluation of KQML as an agent communication language', from Intelligent Agents II — Proceedings of the 1995 Workshop on Agent Theories, Architectures and Languages, Springer-Verlag (1996).

12 Vinoski S: 'CORBA: integrating diverse applications within distributed heterogeneous environments', IEEE Communications Magazine, 14, No 2, pp 133 (February 1998).

13 Kindel C: 'Distributed component object model protocol — DCOM/ 1.0', (January 1998).

14 Visser P R S, Jones D M, Bench-Capon T J M and Shave M J R: 'Assessing heterogeneity by classifying ontology mismatches', in Guarino N (Ed): 'International Conference on Formal Ontology in Information Systems (FOIS'98)', IOS Press, pp 148—162 (1998).

15 Gray P M D, Embury S M, Hui K Y and Kemp G: 'The evolving role of constraints in the functional data model', in JIIS, pp 1—27 (1999).

Nick Fiddian is Head of Department and Professor of Computer Science at Cardiff University. He was educated at London (LSE and ULICS) and Southampton Universities.

His current research interests lie in the field of meta-translation — meta-programmed automatic translation between syntactically diverse but semantically similar programming languages in specific areas of computer application, particularly data/ knowledge-base system interoperation.

Philippe Marti has been a Research Associate on the KRAFT project in the Department of Computer Science at Cardiff University since 1996. He received an Engineering degree in Electronics and Computer Science from the National Engineering School in Brest (France) and was awarded a PhD by the University of Nice (France) in 1996.

His current research is concerned with the use of ontologies for information retrieval purposes and the role of constraints in agent communication languages.

Alun Preece has worked in the area of knowledge-based systems since 1986. He received his PhD from the University of Wales, Swansea, in 1989, after which he spent four years at Concordia University, Montreal, working on a research contract with Bell Canada looking at KBS reliability.

His current research interests are in distributed KBS, software agents and industrial knowledge management.

Jean-Christophe Pazzaglia was associated with the Computer Science Department at Cardiff University in 1997 and 1998 on the KRAFT project. He received an Engineering degree from the National School of Engineering in Computer Science (ESSI) in 1992, and a PhD in 1997 from the University of Nice Sophia Antipolis, France. He is presently employed as a Researcher at Mediatech S.r1., Italy. His current research is based on principles of knowledge communication and is centred around collaborative distance learning — from acquisition and representation to the process and transmission of knowledge among software agents, humans and machines.

Dean Jones has been associated with the Department of Computer Science at the University of Liverpool since 1989. He received a first-class BSc(Hons) from the department in 1992, was awarded a PhD in 1998 and is presently employed as a Research Associate.

His current research includes the use of ontologies in addressing semantic heterogeneity, ontology development, the verification and validation of knowledge-based systems and the formal representation of metaphor.

Kit-ying Hui received his BSc degree in Computer Studies in 1990 from the University of Hong Kong and his MSc degree in Applied Artificial Intelligence in 1994 from the University of Aberdeen (Scotland).

His research interests embrace software agents, databases and constraints.

He is currently pursuing his PhD degree in the Department of Computing Science at the University of Aberdeen.

Zhan Cui received a BSc (1981) and an MSc (1985) in Computer Science from Jilin University in China, and a PhD in Artificial Intelligence from Academia Sinica in 1988. Between 1989 and 1996, he worked as a research fellow for the Universities of Edinburgh and Leeds and the Imperial Cancer Research Fund, and as a lecturer for the Universities of Swansea and Liverpool. He joined BT in October 1996. He has published many papers on mechanical theorem providing, spatial-temporal logic, deductive and object-oriented database, qualitative physics and neural networks. His current research interests include ontology and knowledge management, heterogeneous information integration and agent-based business process management.