Designing for Scalabilty in a Knowledge Fusion System

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

Abstract

The KRAFT project has defined a generic agent-based architecture to support *knowledge fusion* — 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. KRAFT focuses on knowledge in the form of *constraints* expressed against an object data model defined by a shared ontology. KRAFT employs three kinds of agent: *facilitators* locate appropriate on-line sources of knowledge; *wrappers* transform heterogeneous knowledge to a homogeneous constraint interchange format; *mediators* fuse the constraints together with associated data to form a dynamically-composed constraint satisfaction problem, which is then passed to an existing constraint solver engine to compute solutions.

The KRAFT architecture has been designed to be scalable to large numbers of agents; this paper describes the features of the architecture designed to support scalability. In particular, we examine static techniques that underpin the growth of large-scale KRAFT networks, and dynamic techniques that allow reorganisation of a KRAFT network as it increases in scale.

1 Introduction

The KRAFT project (Knowledge Reuse And Fusion/Transformation) aims to define a generic architecture for knowledge fusion. 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.

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).

A key feature of the KRAFT project is that the form of knowledge is restricted to *constraints* expressed against an object data model defined by a shared ontology [7, 13]. This ontology specifies the knowledge available in resources external to the current KRAFT network, and allows the external resource concepts to be expressed in a common internal representation. This internal KRAFT resource is used by *facilitator* and *wrapper* agents and is managed by a special *mediator* agent.

KRAFT builds upon work done in the early 1990s on knowledge sharing and reuse, most notably the results of the Knowledge Sharing Effort (KSE) project [11]. Although it did result in a number of practical applications (for example, [5, 9]), the early work on knowledge sharing and reuse has not had the expected impact in the construction of large-scale, open, distributed knowledge systems. One area that was not addressed in the early work was that of *scalability*: few systems used more than 10 agents. However, there is now a rapidly-growing demand for "Internet scale" systems that support the exchange and processing of rich information in areas such as electronic commerce and knowledge management. Any architecture targeting these areas must be designed with scalability in mind. An important aspect of KRAFT is the use of constraints (intensional data) to reduce the quantities of extensional data being transported and so improve scalability.

This paper can be regarded as a sequel to the paper we presented on the KRAFT architecture at ES99 [13]; here, our focus is on the implementation of the architecture, and in particular on the design features that support scalability. In particular, we will examine *static* techniques that underpin the growth of large-scale KRAFT networks, and *dynamic* techniques that allow reorganisation of a KRAFT network as it increases in scale. Before we can do this, however, we need to review the KRAFT agent-based architecture and its implementation.

The paper is organised as follows. Section 2 presents an overview of the architecture, including the conceptual operations of the main types of agent, and the implementation design. Section 3 examines how scalability has been designed into the architecture as a whole, and the individual agents. Section 4 concludes.

2 The KRAFT Agent Architecture

An overview of the generic 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 knowledge-processing entities, and are described in sections 2.1, 2.2, and 2.3. 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. Within a KRAFT network, the constraints and data are expressed using the concepts defined in the shared ontology.

KRAFT agents communicate via messages using a nested protocol suite. KRAFT messages are implemented as character strings transported by a suitable carrier pro-

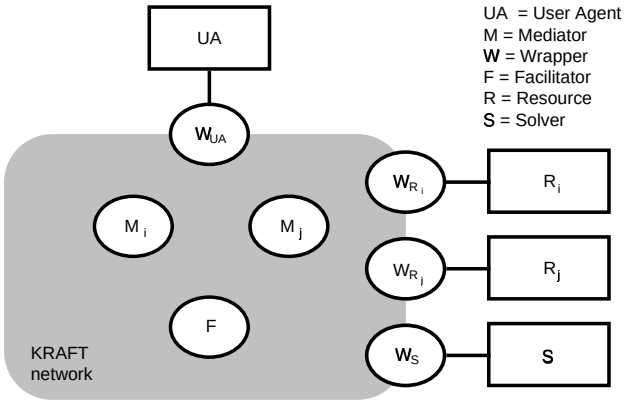

Figure 1: Overview of the generic KRAFT architecture.

tocol. 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 *Constraint Command and Query Language* (CCQL) which is a specialised subset of the Knowledge Query and Manipulation Language (KQML) [10]. Nested within the CCQL message is its content, expressed in the *Constraint Interchange Format* (CIF).

In the current implementation, KRAFT messages are syntactically Prolog term structures. An example message is shown in Figure 2. The outermost `kraft_msg` structure contains a `context` clause (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, a CIF expression (in the implementation, CIF expressions are actually transmitted in a compiled internal format). CIF is described further in Section 2.1.

The following sub-sections examine the operations of each of the three kinds of KRAFT agent in more detail.

2.1 Wrapper Agents

Wrappers are agents that act as proxies for external resources — commonly, these will be "knowledge suppliers". Wrappers serve two purposes: they *advertise* the capabilites of the resource to a facilitator, when the resource comes on-line (see Section 2.3), and they *transform* knowledge between the common interchange format used within a KRAFT network, and the internal format used privately by the resource. In both these tasks they utilise the shared ontology.

2.2 Mediator Agents

KRAFT follows Wiederhold's definition of a mediator as a component that performs a specific information-processing task, often programmed by an individual domain expert [17]; every mediator "adds value" in some way to knowledge obtained from other

```
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"
       ])
   )
```

Figure 2: An example KRAFT CCQL message.

agents. In KRAFT, the main tasks performed by mediators are ontology management and knowledge fusion. To perform knowledge fusion, a mediator gathers constraints from other agents (typically wrappers as described in Section 2.1), and processes them to form a coherent CSP at run-time for solving. This is shown in Figure 3. As part of this task, mediators will typically pre-process the constraints in various ways; they will also need to plan and perform selective database queries as explained in [6].

There will typically be several mediators in a KRAFT network: one to perform each distinct value-adding service. For example, in a KRAFT network performing configuration of PC products, there may be a single configurator mediator, or there may be several configurators, perhaps one for each of several different kinds of PC (laptops, generic desktops, special-purpose workstations, etc) or one for each of several distinct subsystems (CPU, peripheral systems, application software bundles, etc).

2.3 Facilitator Agents

Facilitators are the “matchmaker” agents that allow agents to discover one another [9, 17]. As they come online, agents register their identities, network locations, and advertisements of their knowledge-processing capabilities with a known facilitator. 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.

A resource capability must be represented intentionally and generically for compactness, but in a way that minimises imprecision. The abstract characteristics of a resource are communicated by means of advertisement messages. The terms used in the body of advertisements are defined in the shared ontology. The facilitator encap-

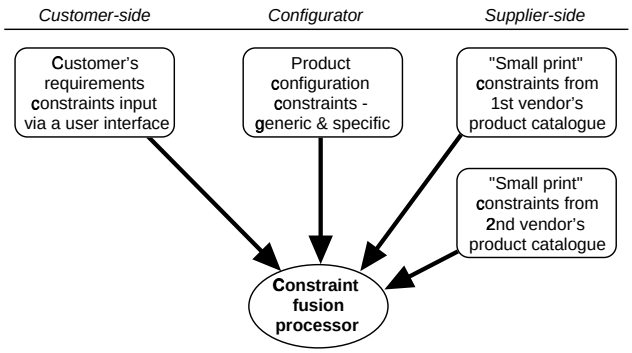

Figure 3: Fusion of constraints from multiple sources.

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

Each KRAFT network requires at least one facilitator. In a large network, there may be multiple facilitators, either for reasons of specialisation or efficiency.

2.4 Implementation of the KRAFT architecture

Inter-agent communication in KRAFT is implemented by message passing using the Linda tuple-space communication model [4]. A Linda server manages the tuple space; clients connect to the space to write or read tuples (messages). KRAFT uses a Prolog implementation of Linda, where tuples are Prolog term structures: instances of the `kraft_msg` term structure shown earlier. To send a message, an agent writes it to a Linda server with the name of the recipient; to receive a message, an agent reads any tuples with its own name as the value of the `receiver` field. An advantage of using this model is that the individual agents do not need to be multithreaded; they choose when to receive any waiting messages synchronously. KRAFT agents can be written in any language provided that they have a Linda client module. Currently, these are available for Prolog and Java agents.

The Linda model is most effective for local-area communication, so to support wide-area KRAFT networks a federated Linda space has been implemented. Each local-area (called a *hub*) has its own Linda server with which local agents interact. The agent namespace has a URL-like *hubname/agentname* syntax. Each Linda server is coupled to a *gateway* agent that relays messages between hubs in a manner similar to an internet router: if a message is posted on the local Linda server with a non-local *hubname* for the recipient, the gateway relays the message to the correct hub gateway agent, which in turn writes it to the hub's local Linda server. This architecture is shown in Figure 4. In the current implementation, the protocol used to carry messages between hubs is TCP via the socket interface; preliminary work has also been done on inter-hub communication using CORBA IIOP [12].

To support debugging, a *Monitor* user agent has been implemented to trace and

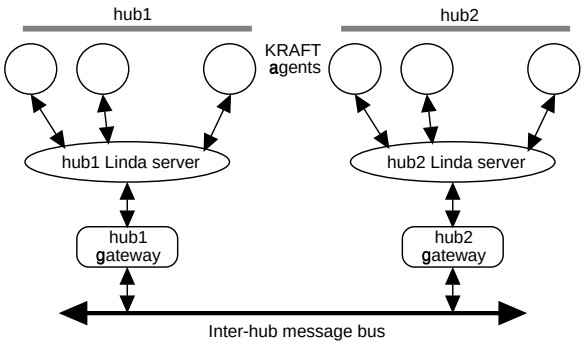

Figure 4: Implementation of the KRAFT architecture.

display the passage of messages across a KRAFT network, shown in Figure 5. Monitor agents are able to register with the gateway agents in order to display activity at non-local hubs, allowing a user to see interactions across the entire KRAFT network.

3 Designing for Scalability in KRAFT

During the design of the KRAFT architecture, scalability was borne in mind. The issue of scalability in a distributed environment is concerned with reducing the degradation in performance as the work load increases. This work load can be measured in a number of ways, such as the number of users, activities, resource providers, and sites. Inevitably as the distributed environment grows there is some degradation in performance. Broadly, there are two kinds of technique that can be used to tackle this problem: *static* and *dynamic* techniques.

- *Static techniques* are generally concerned with architectural decisions that have been taken to support the general scalability of the system and are independent of the dynamic situation.
- *Dynamic techniques* monitor the current state of the distributed system (for example, the demand on each resource, or the performance profile of each agent), and attempt to respond to fluctuating performance in different parts of the system by ameliorating the factors causing performance degradation. Hence, dynamic techniques respond to feedback from monitors which are evaluating the current performance to identify peaks and/or bottlenecks.

In KRAFT, both static and dynamic of scalability techniques have been employed. To this end, six strategies were built-into the architecture:

1. An intelligent strategy for resource querying and constraint solving aims to minimise number of messages, and volume of data transferred, conserving bandwidth.
2. The federated messaging architecture (internet-style) using Linda hubs aims to minimise network traffic, while also being easy to manage and fault-tolerant.

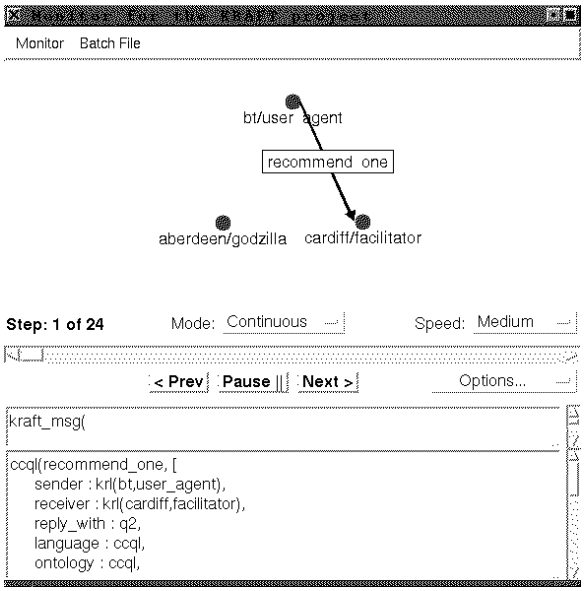

Figure 5: Screenshot of the KRAFT Monitor user agent.

3. Mediators can easily be replicated on their local hub, allowing for concurrent processing and load-balancing.

4. Use of virtual machines for agent implementations (e.g. Java and Prolog) are encouraged, to allow agent mobility in cases where processing needs to be moved local to a resource, or where it is necessary to transfer/replicate an agent on a different hub.

5. Facilitator bottlenecks are avoided by allowing for facilitator replication and specialisation (for example, having a hierarchy of facilitators, similar to DNS), and by using generic advertising to reduce the number of advertisements.

6. The shared ontology supports ontology clusters which in turn allow for evolvability (different worlds) and bottom-up integration of independently-created networks.

All of these strategies have a static aspect, in that they allow a KRAFT network designer to address the problem of scalability at design-time. However, strategies (3–5) have a significant dynamic aspect, as described below.

We will now examine each of these strategies in more detail.

3.1 Querying and Solving Strategy

As described in Section 2.2, mediators that perform constraint fusion undertake a combination of operations in constructing a CSP at run-time: gathering and pre-processing

a set of constraints from contributing agents, and querying database wrappers in order to populate the domains for the CSP. The two most costly operations here are actually external to the mediator: these are:

1. the network traffic generated by the constraint and data gathering process; and
2. the constraint solving process carried out by the wrapped solver to which the mediator dispatches the CSP.

The KRAFT architecture is designed to try to minimise both these costs, as follows:

1. In terms of network traffic, constraints are a compact way of transmitting what could otherwise be substantial volumes of data. In a conventional information interchange approach, data instances would be transmitted extensionally, whereas in the constraint approach, each constraint is an intensional definition of a potentially-large amount of data [15].

   Moreover, where it *is* necessary to transfer a number of instances extensionally (for example, to populate a variable domain for a CSP), constraints can be used by the mediator as a *filter* to reduce the number of instances based on what is known about the solution space so far. For example, if it is already known that a PC customer needs at least a 17-inch monitor, there is no point in the monitor vendor shipping instances of 15-inch monitors. The "screen size $>=$ 17 inches" constraint would be sent by the mediator to the monitor vendor's wrapper for it to use as part of the internal query on the monitors product catalogue.

2. In terms of constraint solving, the mediator can perform an arbitrary amount of pre-processing on the CSP before sending it to the solver. It can also select what it deems to be the optimal kind of solver for the problem-at-hand. The pre-processing can include simplifying the problem (for example, if redundant constraints are detected) or even aborting the solving process (for example, if a conflict is detected in the CSP that renders it insoluble). Also, the aforementioned process used to build the CSP in terms of gathering instances to populate the variable domains will itself tend to simplify the CSP by filtering out a significant number of "useless" values (this is similar to the forward-checking technique used in constraint solving [8], but applied in the distributed context).

These normally lead to a reduction in data being transported in the network, and so improve scalability.

3.2 Federated Messaging Architecture

Unlike some agent communication architectures (for example, JATlite[1]) KRAFT does not rely on a centralised message routing mechanism. Routing in KRAFT, like routing in the Internet, is decentralised. The Linda hub architecture shown in Figure 4 allows messages local to a hub to pass unnoticed by other hubs. When the hub is entirely local

[1] `http://java.stanford.edu/`

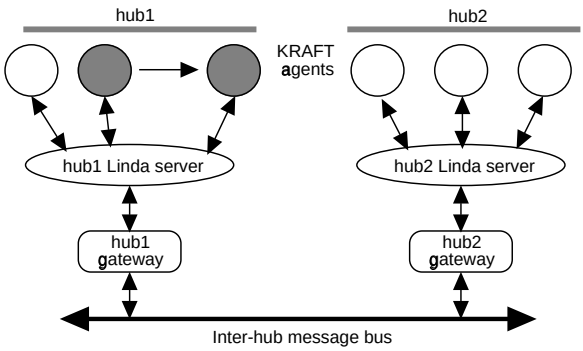

Figure 6: Cloning a mediator on a local hub in a KRAFT network.

to a single host, no network traffic at all is generated (messages are passed through the TCP/IP "loopback" interface). The only wide-area network traffic consists of inter-hub messages, which pass between widely-separated gateways. Section 3.4 explains how even this traffic can be minimised.

An important exception here is where monitor agents are configured to monitor activity at more than one hub. In this case, the hub message server is notified to "echo" messages between monitored hubs, and this can lead to substantial network traffic. Therefore, such monitoring is recommended only for testing, and never for use in an operational KRAFT network.

3.3 Mediator Replication

It is conceivable that mediators whose services are much in demand will become bottlenecks. Such bottlenecks are easily detected by the hub servers, which can see that message traffic to and from particular agents is greater than normal, or are not being processed as rapidly as would be desirable. (This is like an office manager observing the "in" and "out" trays of workers in an office: in KRAFT these "trays" are collections of messages on the Linda tuple space managed by the hub server.) Locally, it is relatively easy to clone KRAFT mediators, because these agents do not tend to have persistent storage (if they do, they will typically use a shared private database for this purpose). Therefore, a mediator can be cloned in principle by simply asking the local OS to spawn another instance of the mediator process. The mediator would be passed a unique agent name by the hub server. This is illustrated in Figure 6 — the shaded agent (a mediator) is cloned on *hub1*.

In this way, mediator services can easily be replicated locally to provide concurrent processing. This technique is equally useful for replicating ontology-management mediators as well as those performing knowledge fusion.

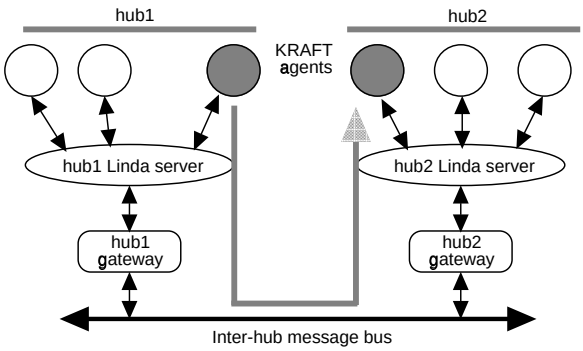

Figure 7: Replicating a mediator on a remote hub in a KRAFT network.

3.4 Agent Mobility

Section 3.3 covers the case where a mediator is replicated locally. It is also possible to replicate a mediator on a different hub. (Or indeed, to relocate the mediator, by replicating it then removing the original.) This is possible when the mediator is implemented in a language that uses a virtual machine environment. KRAFT encourages the use of such languages; currently, the trial systems have been implemented using Prolog and Java. In these cases, the code for the mediator can be encapsulated in an inter-hub message directed to a hub manager. Upon receipt, the hub can spawn an instance of the appropriate virtual machine and pre-load it with the encapsulated agent. This is illustrated in Figure 7 — the shaded agent (a mediator) is replicated on *hub2*, after being shipped over the network from *hub1*.

This kind of mobility is most advantageous when one wants to move the processing capability closer to the data. Localised communication is always far less costly than wide-area communication, so moving a mediator from a remote hub to the same hub as the data resources can improve performance significantly.

3.5 Facilitator Federations

As the scale of a KRAFT network grows, there is a danger that facilitators will become bottlenecks. A number of strategies are available to avoid this problem.

Normally, each hub will have its own local facilitator, to which requests can be directed without the cost of wide-area traffic. If each of these facilitators is to have knowledge of the entire multi-hub KRAFT network, then they obviously must exchange advertisements among one another. However, recommend requests are typically much more common than advertise requests, so most of their work then will involve local message traffic.

If the network becomes too large to allow each local facilitator to store complete advertisement information, then there are a number of possibilities to reduce the amount of information held by each facilitator. One option is to have *specialist* facilitators, each holding knowledge about a particular area of the shared ontology (for example, in a consumer electronics marketplace, there could be a facilitator that

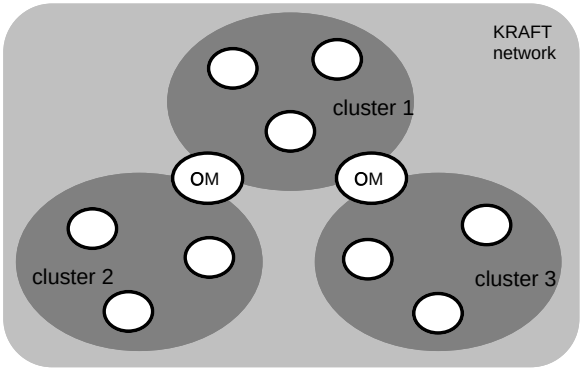

Figure 8: Three ontological clusters within the same KRAFT network.

knows about the PC sector, another that knows about the mobile phone sector, and so on). A "meta-facilitator" would be consulted first to locate the appropriate specialist facilitator for a given request. Of course, any of these facilitators could also be replicated if necessary. Another possibility is to partition the facilitation space according to location, where each facilitator would have knowledge about its local hub (or local hubs), and would direct requests that can't be answered locally to a neighbouring facilitator. This approach is essentially similar to the Internet routing scheme.

The use of generic advertisements will reduce the number of advertisements as it eliminates the need to hold specific advertisements. It also reduces the number to be examined when locating appropriate external resources, albeit at the expense of a possibly more complex matching algorithm.

3.6 Ontology Clusters

The schemes outlined in Section 3.5 cover only the cases where the entire KRAFT network uses a single shared ontology, against which all advertisements and requests are expressed. CCQL allows for mutliple ontologies to coexist within the same (using the `ontology` field in each message) but, more importantly, allows for agents using different ontologies to interoperate by means of *ontology clusters* and inter-ontology mappings. An ontology cluster is a community of KRAFT agents that use the same shared ontology; a cluster that uses shared ontology SO_1 can interoperate with a cluster that uses shared ontology SO_2 if and only if there is an *ontological mapping* from SO_1 to SO_2. Messages with content expressed against SO_1 would be sent to a designated *ontological mediator* (OM) that would transform the content to SO_2 before fowarding the message to the intended recipient. Figure 8 shows three ontological clusters within the same KRAFT network, bridged by ontological mediators.

This part of the design for the KRAFT architecture is elaborated in [14]. The ontological clusters allow for evolvability (different worlds which must be made to inter-operate) and bottom-up integration of independently-created KRAFT networks.

4 Conclusion

Addressing the issue of scalability in a distributed environment involves managing the potential degradation in performance as work load increases. Typically, this is done by employing a combination of static (design-time) and dynamic (run-time) strategies. In KRAFT architecture, the static techniques available to support scalability are as follows:

- the use of intensional data reduces the quantity of extensional data being transported in the network;
- the use of generic advert representation reduces the number of adverts stored and analysed;
- the use of optimal solvers and filters reduces the amount of extensional data accessed from external sources in CSP;
- the ability to clone at all KRAFT sites the facilitator, mediators and ontology mean that it is possible to design the architecture so that there is no network traffic if a CSP can be solved with local facilities only;
- the Linda hub design means that local messages are processed locally — they also support an easy implementation of monitors which trigger dynamic techniques as the tuples held in a Linda hub show the dynamic state of many of the KRAFT components; and
- the use of ontology clusters and a build-up approach to creating the local shared ontology for a cluster means they reflect their user community needs, which again restricts the number of occasions when there is a need to link to other ontologies in the cluster.

The dynamic techniques that support run-time scalability in the KRAFT architecture are founded on the use of a managed, hub-based design: the messages at the Linda hubs can be monitored to determine where the bottlenecks are occurring and which agents are involved. As a result of this monitoring, agents can be cloned at the same or another site to reduce network traffic and overall time when an agent is overloaded. Agents can migrate to new sites if the platform configuration at that site and/or the current situation is better suited to their processing needs.

We contend that these static and dynamic features mean that the degradation of performance in a KRAFT network as workload increases is graceful, and the environment is able to react to fluctuation in usage patterns if this is needed.

Acknowledgements. KRAFT is a collaborative research project between the Universities of Aberdeen, Cardiff and Liverpool, and BT. The project has received funding from the UK Engineering and Physical Sciences Research Council, and BT.

References

[1] J. Andreoli, U. Borghoff, and R. Pareschi, Constraint agents for the information age, *Journal of Universal Computer Science*, 1:762–789, 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] R. Bayardo et al, InfoSleuth: agent-based semantic integration of information in open and dynamic environments, In *Proc. SIGMOD'97*, 1997.

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

[5] M. Cutkosky, R. Engelmore, R. Fikes, M. Genesereth, T. Gruber, W. Mark, J. Tenenbaum, and J. Weber, PACT: an experiment in integrating concurrent engineering systems, *IEEE Computer*, 26:8–27, 1993.

[6] 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.

[7] P. Gray, A. Preece, N. Fiddian, W. Gray, T. Bench-Capon, M. Shave, N. Azarmi, and M. Wiegand, KRAFT: knowledge fusion from distributed databases and knowledge bases, in *Proc. 8th International Workshop on Database and Expert System Applications (DEXA-97)*, pages 682–691, IEEE Press, 1997.

[8] P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, 1989.

[9] 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.

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

[11] 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.

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

[13] A. Preece and K. Hui and W. A. Gray and P. Marti and T. Bench-Capon and D. Jones and Z. Cui, The KRAFT Architecture for Knowledge Fusion and Transformation, In *Research and Development in Intelligent Systems XVI (Proc ES99)*, Springer, pages 23–38, 1999.

[14] M. Shave, Ontological structures for knowledge sharing, *New Review of Information Networking*, 3:125–133, 1997.

[15] M. Torrens and B. Faltings, Smart clients: constraint satisfaction as a paradigm for scaleable intelligent information systems, In T Finin and B Grosof (eds) *Artificial Intelligence for Electronic Commerce: Papers from the AAAI-99 Workshop*, AAAI Press, 1999.

[16] 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, pages 148–162, 1998.

[17] G. Wiederhold and M. Genesereth, The basis for mediation, In *Proc. 3rd International Conference on Cooperative Information Systems (COOPIS95)*, 1995.