An Open Framework for Cooperative Problem Solving

Mauro Gaspari Enrico Motta, and Arthur Stutt

Technical Report UBLCS-95-5

January 1995

Laboratory for Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)
The University of Bologna Laboratory for Computer Science Research Technical Reports are available via anonymous FTP from the area ftp.cs.unibo.it:/pub/TR/UBLCS in compressed PostScript format. Abstracts are available from the same host in the directory /pub/TR/UBLCS/ABSTRACTS in plain text format. All local authors can be reached via e-mail at the address last-name@cs.unibo.it.

### UBLCS Technical Report Series

93-7  *Distributed Programming with Logic Tuple Spaces*, by P. Ciancarini, April 1993.
93-8  *Coordinating Rule-Based Software Processes with ESP*, by P. Ciancarini, April 1993.
93-14  *Guard Compilation in Logic Shared Dataspaces Languages*, by M. Garpari, June 1993.
94-6 Exploring the Coordination Space with LO, S. Castellani, P. Ciancarini, April 1994.
94-7 Comparative Semantics of LO, S. Castellani, P. Ciancarini, April 1994.
94-20 Dynamic Allocation of Signature Files in Multiple-Disk Systems, P. Ciaccia, August 1994.
94-21 Parallel Independent Grid Files Based on a Dynamic Declustering Method Using Multiple Error Correcting Codes, P. Ciaccia, November 1994.
An Open Framework for Cooperative Problem Solving

Mauro Gaspari, Enrico Motta, Arthur Stutt

Technical Report UBLCS-95-05
January 1995

Abstract

Hybrid, knowledge-based systems comprise a number of heterogeneous agents, which make use of different knowledge representation languages. The VITAL-KR is a software architecture providing the basic sub-structure for integrating a number of software and human agents which cooperate during problem solving. In this paper we give an overview of the architecture of the VITAL-KR and show how it supports the development of hybrid applications, comprising both human and heterogeneous software agents, and the integration of pre-existing software modules into an application. The VITAL-KR provides a number of advantages over alternative AI programming environments, which make it well-suited for the development of hybrid, knowledge-based applications. Its communication primitives are generic, as they do not depend on the structure of a particular knowledge representation system; it is extendible, as it provides well-defined mechanisms for integrating new software modules; it enjoys a formal, unambiguous specification; it comprises mechanisms to ensure the consistency of the overall, hybrid knowledge base; and it provides a uniform mechanism for integrating both software and human agents.

1 This research was part-funded by the Commission of the European Communities ESPRIT-II Project 5365 - VITAL. The partners in the VITAL project are the following: Syseca Temps Reel (F), Bull Cediag (F), Onera (F), The Open University (UK), University of Nottingham (UK), University of Helsinki (SF), and Andersen Consulting (E). Mauro Gaspari has been partially supported by the Italian Ministry of Scientific and Technological Research and Universities (MURST) and by the ESPRIT BRA Project 9102- Coordination.
2 Dipartimento di Scienze dell'Informazione, Universita’ di Bologna, Piazza S. Donato 5, Bologna, Italy. E-mail address is gaspari@cs.unibo.it
3 Knowledge Media Laboratory, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK, email: e.motta@open.ac.uk, a.stutt@open.ac.uk
1 Introduction

When designing complex, knowledge-based applications a knowledge engineer first identifies the tasks which need to be tackled and then, if the application is sufficiently complex, assigns these tasks to the different agents - human or software - which are capable of carrying them out. Human and software agents in a cooperative problem solving scenario can be characterized in terms of the role they play in problem solving (e.g. explanation, classification, translation, control). Software agents can also be described in terms of the type of architecture (e.g. conventional vs. knowledge-based software agents) and the representation and inference paradigm employed (e.g. rule-based components vs. frame-based representations). Knowledge-based applications comprising a number of cooperating software agents which make use of diverse representation and inference paradigms are said to be hybrid. Over the past few years there has been a lot of interest in hybrid systems, for a number of reasons.

- Many current knowledge-based applications are very large and complex and it is important to be able to reuse pre-existing software modules by integrating them in the application. These modules might make use of alternative representation and inference paradigms and it is therefore necessary to devise cooperation mechanisms which can support the interoperability of heterogeneous inference modules.

- There is now a widespread consensus in the AI community that there is no magic knowledge representation bullet, i.e. "there is no single knowledge representation that is best for all problems, nor is there likely to be one" [Nec91]. Therefore, researchers are developing environments which can offer the user ‘more choice’ in terms of AI programming paradigms. Such environments can be viewed as providing software support for developing hybrid, knowledge-based applications. For instance, the CYC system [Len90] comprises a few dozen inference mechanisms (the heuristic level) which are integrated by means of a homogeneous representation (the epistemological level).

- The radical improvements which have occurred over the past few years in the area of communication and networking has made it more advantageous to develop distributed applications characterized by a number of software agents residing on a number of remote hosts and making use of diverse representations.

The VITAL-KR is a software architecture providing the basic sub-structure for developing hybrid, knowledge-based applications, comprising a number of software and human agents which cooperate during problem solving. The VITAL-KR provides implementation-level support in the VITAL workbench [Dom93], a methodology-based workbench covering the whole life-cycle of knowledge-based system (KBS) development, from requirements specification to implementation. The VITAL-KR allows the construction of modular KBS, facilitates the reuse of KBS modules, provides standard protocols for integrating software or human agents into an application, and supports experimentation in the design of hybrid KBS.

In this paper we give an overview of the VITAL-KR, focusing in particular on its architecture and the support it provides for developing applications comprising human and heterogeneous software agents - i.e. software agents which make use of diverse representation and inference paradigms.
2 Approaches to Hybrid Architectures

Commercial, hybrid knowledge engineering environments such as KEE™ and KnowledgeWorks™ are limited in terms of the numbers of paradigms they support (usually just rules and frames). More importantly, they are not easily extendible to embed alternative inference mechanisms. In the words of Daniel Corkill, "AI shells have a Ptolemaic view of their universe, most cannot be easily integrated as closely-coupled components of a larger problem-solving system." [Cor91].

In the GBB-OPS5 architecture [Cor91] it is possible to extend the reasoning mechanisms (at least in theory) by the specification of new types of GBB knowledge sources. While systems of this type are therefore extendible, they remain embedded in that the communication primitives and the integrating structure are kernel-specific. This means that even if a number of inference engines are integrated into the GBB framework, as far as the programmer of the resulting hybrid environment is concerned, the architecture remains that of a blackboard (albeit of an extended sort).

Embedded architectures are currently the principal way in which hybrid programming environments are defined. The main problem with embedding as the integration technique for hybrid systems is that the resulting cooperation between the two reasoners depends on the design of the kernel, rather than on a generic and principled cooperation protocol. That is, because the emphasis is on integrating the component tools with the kernel, only kernel-dependent mechanisms for cooperation are provided. This aspect limits both the generality and the extendibility of these approaches.

In contrast to the aforementioned programming environments, much of the work on distributed AI - for instance HECODES [Zha91] - is characterized by the adoption of a non-embedded, more extendible approach, where the system components are 'democratically organized' (i.e. there is no component in which all the others are embedded). Current work on a standard for communicating agents, KQML [Fin94], also presupposes a loose, ‘horizontal’ kind of organization. These non-embedded architectures do not suffer from the limitations we ascribe to embedded approaches. Because all components cooperate at the same level, and are only loosely integrated, they are more suitable than embedded ones for supporting the integration of pre-existing, heterogeneous software modules. Moreover, because the control and communication aspects of a non-embedded architecture do not depend on the embedding kernel, the communication language used by the components of the hybrid application can be truly generic and diverse cooperation strategies can be supported.

The VITAL-KR is a non-embedded architecture which uses a horizontal extendible organization to support the cooperation between a number of software and human agents during problem solving. The adoption of a non-embedded organization allows us to define a generic - implementation-independent or knowledge-level [Gas94] - communication language, and enables users/designers to experiment with a range of configurations. Moreover, it also enables us to specify a simple and general framework for integrating additional software modules, thus supporting implementation-level reusability. This general framework also supports the integration of human agents in a cooperative problem solving scenario, without resorting to special-purpose mechanisms. At the knowledge level there is no difference between software and human agents in a VITAL-KR application.

Because non-embedded approaches only afford loose integration of components, it is normally difficult for this class of architectures to guarantee the consistency of a hybrid KBS, and to describe its behaviour formally. In the design of the VITAL-KR we have addressed these two problems by i)
defining the communication primitives so that it is possible for a justification-based truth maintenance system (JTMS) [Doy79] to monitor the consistency of the integrated, hybrid knowledge base (KB); and by ii) providing a formal account of the architecture. Because of space limitations these aspects won’t be discussed here and the interested reader is referred to [Gas93].

3 The VITAL-KR
The VITAL-KR ‘shell’ consists of six main (sets of) components: i) an Inference Scheduler (IS), which is the main control and communication component; ii) a set of default inference engines which provide the VITAL-KR user with a range of representation and inference capabilities; iii) a set of default visualization/acquisition (V/A) cliches, which provide mediating interfaces between a human agent and a VITAL-KR application; iv) a set of default Knowledge Exchange Interfaces (KEIs) which define how VITAL-KR generic communication primitives are to be translated into the specific ones used by software or human agents; v) a Meta-Knowledge Repository (MKR), which contains the information used by the IS for dispatching messages; and vi) a JTMS, which can be used to ensure the consistency of the overall hybrid KB, and/or to provide default truth maintenance facilities to those engines which do not implement their own TMS.

Figure 1. The architecture of a generic VITAL-KR application

An application developed by means of the VITAL-KR ‘shell’ consists of a number of Inference Modules (IMs) - see figure 1. Each IM is an internally homogeneous component of the overall hybrid KB and is therefore associated with a particular inference engine or V/A cliche. IMs in a VITAL-KR based application communicate through a functional interface [Lev84] which defines the range of messages that can be exchanged by the IMs.
Figure 2 shows the internal organizations of human and software agents in a VITAL-KR application. The human agent in the figure - IM1 - interacts with the application through a V/A interface which is either a customization of one of the V/A cliches provided in the VITAL workbench, or a special-purpose one developed using the VITAL software support for defining V/A interfaces [Dom93]. This includes extensive support for developing graphical editors, domain-level visualizations and knowledge acquisition interfaces. In particular the Viz tool [Dom93] provides a very high-level language to develop domain-level visualizations, based on the concepts of events, players, mappings, and navigators.

A software agent (e.g. IM2) is basically a homogeneous KB, which uses one particular inference engine, selected from the VITAL-KR library of inference engines. This library currently comprises half a dozen engines including: a frame-based language; a task scheduler; a Prolog engine; and an OPS5 implementation.

As shown in the figure, communication from and to an IM is mediated by the associated KEI. This aspect will be discussed in detail in sections 3.3 and 3.4.
3.1 Communication in the VITAL-KR: the Functional Interface

The functional interface plays three roles. First, it defines the range of messages that can be exchanged by the IMs. Second, it defines at a black-box level the range of functionalities that the VITAL-KR can provide. Third, because the VITAL-KR can only provide the functionalities supported by its components, it provides a functional framework for integrating IMs and for adding new engines to the engine library, and new V/A cliches to the cliche library. Thus, it provides a non-ambiguous specification for those VITAL-KR users who wish to integrate additional engines or V/A cliches. The functional interface does not depend on a particular configuration of the VITAL-KR, nor on the internal structure of an IM. This makes it ideal for integrating heterogeneous software agents as well as human ones.

The functional interface supports a number of classes of generic operations, which include support for entering new data (TELL); retracting existing data (RETRACT); querying the VITAL-KR (ASK); connecting and disconnecting inference modules (CONNECT); and interacting with the JTMS.

The primitives in the functional interface have been chosen by following two main criteria. On the one hand we want to have a set of primitives which enable us to deal with IMs as black boxes. In other words, “they are characterized not in terms of the structures they use to represent knowledge, but in terms of what they can be asked or told about some domain” [Lev84]. On the other hand, because the aim is to provide a practical testbed for building large, hybrid KBSs, a more fine grained breakdown has been used than would have been strictly necessary from a logical point of view. This can be seen in the case of the ASK class of primitives, which distinguishes between requests that are simple data retrieval operations (fetch, fetchall), and those which may require IMs to perform inferences (prove, proveall, establish, establishall).

All communication within, from, and to the VITAL-KR is centralized through the IS, and mediated by the functional interface. Messages have the structure: \(<\text{functional-interface-command}\> \,<\text{c-expression}\>). A c-expression (or expression in the common language) is of the form: \(<\text{predicate}\> \,<\text{arg1},...<\text{argn}\>\rangle\), with \(n \geq 1\). The predicate of a c-expression is a lisp symbol, and the optional arguments are arbitrary Lisp S-expressions which can contain logical variables. A ground c-expression (i.e. one which does not contain variables) is called a g-expression. The predicate in a c-expression defines its type.

To take part in the problem solving process IMs must be connected to the IS using the CONNECT primitives which are part of the functional interface. These are also used to declare which message types an IM is interested in.

In the case of TELL, ASK, and RETRACT operations, the type of the c-expression is used by the IS to decide which IMs can deal with the operation in question. The advantage of having a simple type-based dispatching mechanism is that it minimizes the dispatching overhead. The mapping from types to IMs is stored in the MKR, which also contains knowledge of how to order the dispatching when more than one IM can deal with a particular message.

The IS associates an agenda with each IM and the JTMS, and uses these to manage the flow of messages to and from each of these components. Each agenda stores all messages dispatched to the corresponding component but not yet processed. Before a message is processed by a particular IM,
its agenda is cleared - i.e. all pending messages are executed. This is done to ensure that the query is processed in an up-to-date environment.

This centralized model of communication (i.e. IS-mediated) guarantees a more modular architecture, as both the communication between the VITAL-KR and the external world, and the communication between components of the VITAL-KR no longer depend on the particular configuration of the system. A particular IM need not ‘know’ which IM (if any) can deal with a message which it originates. IMs send generic messages to the IS; this determines those which can deal with them, and dispatches concrete messages to the interested IMs.

An important aspect of the specification of the functional interface is that, at a black box level, there is no difference between generic and concrete operations, for example between a generic and a concrete prove. The only difference is that generic messages operate on a virtual hybrid KB, while concrete ones operate on an IM-specific one. In the rest of the paper, we’ll prefix generic messages with “c-”, to indicate the associated concrete message.

The functional interface operations are summarized in table 1.

<table>
<thead>
<tr>
<th>TELL</th>
<th>Assert, AssertInfer, Modify</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASK</td>
<td>Fetch, Fetchall, Prove, Proveall, Establish, Establishall</td>
</tr>
<tr>
<td>RETRACT</td>
<td>Unassert</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Define-IM, Integrate-IM, Unintegrate-IM, Interested-IM, Uninterested-IM</td>
</tr>
<tr>
<td>JTMS</td>
<td>NewJ, RemoveJ, Justify.</td>
</tr>
</tbody>
</table>

Table 1. The functional interface.

3.2 Defining a KEI for a Software Agent

A KEI specifies how an IM handles incoming and outgoing messages, i.e. it provides a bi-directional mapping between generic functional interface operations and IM-specific ones. Although it is possible to define IM-specific KEIs, in general a software IM ‘inherits’ its KEI from the associated inference engine. For instance, the definition below shows the definition of the KEI associated with an OPS5 inference engine.

```
(vkr-define-KEI 'ops
 :vital-functions (translate-logical-pathname
   "vitalkr:engines;ops-int;vital-from-ops")
 :c-functions (translate-logical-pathname
   "vitalkr:engines;ops-int;ops-concrete")
 :package 'ops)
```

Figure 3 shows part of the file ops-concrete, which specifies how the primitives in the VITAL-KR functional interface are specialized for the OPS5 engine.
The definitions in Figure 3 are based on the specification of the functional interface primitives which are given in Section 3.1. The concrete assert operation for OPS5 corresponds to the built-in make which adds an element to the OPS5 working memory. The concrete prove operation runs the system and checks at the end of the recognize-act cycle if the query c-expr appears in the OPS5 working memory.

Figure 4 shows part of the contents of the file vital-from-ops containing the definitions of the VITAL-KR primitives to be used in the right hand side (RHS) of OPS5 production rules. The definitions of the ASK primitives (vkr-fetch, vkr-prove and vkr-proveall) show that two steps are needed: first the built-in VITAL-KR function is called and then the query is asserted into the OPS5 working memory with variables instantiated as given by the answers from the VITAL-KR. In the successive cycle the asserted fact will be available for further deduction.
3.3 Defining a KEI for a Human Agent

Human agents in the VITAL-KR are represented as IMs. Therefore they interact with other human and/or software IMs by means of the functional interface operations. Consistently with the case of software IMs, KEIs associated with human agents are also normally defined for generic V/A cliches, rather than with an actual cliche instantiation. As an example we show here the integration of a notepad-style interface into the VITAL library of V/A cliches. The notepad stores assertions and provides a dialogue interface to query the VITAL-KR. Such a KEI is defined as follows.

(vkr-define-KEI 'notepad
  :c-functions (translate-logical-pathname
    "vitalkr:vizacq;user;note-pad;note-pad-concrete")
  :package 'note-pad)

This definition only specifies how this KEI handles incoming messages; outgoing messages (from the notepad to the VITAL-KR system) simply call the VITAL-KR functional interface commands.
The concrete functions which specify how the notepad V/A cliche handles a subset of the VITAL-KR incoming messages are shown in figure 5.

```lisp
;;; ASSERT = insert into the notepad
(defun c-assert ((to notepad) g-expr &key j receivers)
  (declare (ignore j receivers))
  (dr-add (notes to) g-expr)
  (refresh (window to)))

;;; UNASSERT = remove from the notepad
(defun c-unassert ((to notepad) c-expr &key receivers)
  (declare (ignore receivers))
  (let* ((notes (notes to))
          (env (match1 notes c-expr)))
    (unless (eq env :fail)
      (dr-remove notes (instantiate c-expr env))
      (refresh (window to))))

;;; FETCH = check the notepad, this is an automatic answer
(defun c-fetch ((to notepad) c-expr &key receivers)
  (declare (ignore receivers))
  (match1 (notes to) c-expr))

;;; PROVE = ask the user
(defun c-prove ((to notepad) c-expr &key receivers)
  (declare (ignore receivers))
  (let* ((notes (notes to))
         (env (match1 notes c-expr)))
    (when (eq env :fail)
      (get-answer (c-expr-to-string c-expr))))
```

Figure 5. Concrete messages for a note-pad.

The concrete assert operation (c-assert) stores a fact in the notepad while its inverse (c-unassert) removes it. When it is modified the notepad is re-displayed to the user. The concrete fetch operation (c-fetch) is automatic and does not require the user to give an answer; it simply looks for a c-expression in the notepad which matches c-expr. Finally, the concrete prove operation (c-prove) looks for c-expr in the notepad and, in case of failure, opens a dialogue box which allows the user to give an answer to the query (perhaps after he/she has sent other queries to the VITAL-KR by means of functional interface operations).

### 3.4 Defining Software and Human IMs

A software IM is defined by specifying its name, its inference engine, its knowledge base, and two lists of predicate names: the query dispatching directory (qdd-list) and the data dispatching directory (ddd-list). These indicate respectively the types of queries which the IM is able to answer and the types of data which the IM is interested to.

A human IM is defined by means of the same mechanism, i.e., by specifying a name, a V/A cliche, eventual additional user interface software, and the dispatching directories. Thus, the dispatching directories indicate the types of knowledge which a user can contribute as well as the sort of events
about which he/she must be informed. For instance, in our solution to the SisyphusI room allocation problem (Motta et al., 1994), we used Viz to develop an interface which allows the user to visualize the room allocation process, see Figure 6. This interface is only one-way, i.e. it allows the user to visualize the problem solving process, but he/she cannot interfere with it. Once developed, the interface was stored in our library of V/A cliches, a user IM was created and integrated in the application. This integration was trivially accomplished by declaring that the user IM only had to be informed about allocation events.

\[ \text{Figure 6. Visualizing the room allocation process} \]

4 Extendibility in the VITAL-KR System

Additional inference engines can be added by supplying a KEI which specifies how the new engine handles (a subset of) the functional interface operations, i.e. how to translate a generic communication primitive into an engine-specific operation. Thus, defining a new engine in the VITAL-KR actually means specifying the KEI, i.e., the way in which the engine in question interacts with the VITAL-KR.

An engine can only be integrated in the VITAL-KR if it satisfies the following requirements:

---

4 This interface was developed by John Domingue
1. *An engine must support (a subset of) the Functional Interface.* This is an obvious requirement. However, sometimes it can be difficult to decide how a particular functional interface primitive should be instantiated in a particular engine. For this reason we provide a formal, non-ambiguous specification of the functional interface (Gaspari et al., 1993).

2. *An engine must support multiple instantiations of itself.* This is a very important requirement needed to enable the VITAL-KR user to map multiple IMs to the same engine, and to enable IMs to deal with multiple, recursive queries. This capacity for suspension without blocking allows cooperation at the knowledge level (Gaspari and Motta, 1994).

3. *An engine must be able to do IO in terms of c-expressions.* That is to say that it must either use c-expressions internally or be able to translate from c-expressions to its internal formalism and vice versa. This is not a very limiting requirement, as the large majority of AI tools use symbolic representation. However, in some cases some programming work is needed to ensure smooth translation.

This set of requirements also apply to V/A cliches. In practice however, these tend to be less complex than inference engines and therefore they are much easier to integrate.

5 Related Work

**Spark, Burn, FireFighter.** The Spark, Burn, FireFighter (SBF) framework [Kli93] consists of a set of tools that allow a development team to i) analyse a workplace, ii) identify the tasks which can be efficiently automated, iii) select and configure appropriate solutions for these tasks, and iv) modify the workplace allowing the integration of the newly-defined software agents, so that they can cooperate with human agents. The strength of this approach is that it combines software support for workplace analysis with mechanisms supporting software reuse, integration, and the generation of knowledge acquisition interfaces. In contrast with SBF the role of VITAL-KR is to afford implementation-level support within the VITAL workbench by providing an architecture to support the development of hybrid systems, as well as cooperation between human and software agents. That is, VITAL-KR only covers the implementation stage of the knowledge engineering life-cycle. However, the VITAL workbench does provide a number of facilities which are similar to those provided in the SBF framework. For instance, VITAL comprises a library of reusable problem solving models at the knowledge level, which can be used to drive the knowledge acquisition process [Mot94], while the VITAL-KR supports implementation-level reuse. As we have seen, the VITAL workbench also supports the generation of visualization and acquisition interfaces. It seems to us therefore that the main difference between VITAL and SBF is more one of focus than one of approach. SBF focuses particularly on the issue of analysing and integrating human and software agents in the workplace. VITAL provides a general-purpose methodology and workbench for KBS development, which is also suitable for deploying hybrid applications - comprising both software and human agents - in the workplace.

**Intelligent Agent Framework.** The Intelligent Agent (IA) framework for enterprise integration [Pan91] provides the basis for integrating people and computer systems in large manufacturing enterprises. Human agents cooperate with a large number of intelligent software agents which automate some task or job function. The IA framework provides a general and pragmatic approach
to cooperative distributed problem solving which aims to enable “end-users to program simple agents that do useful things for them” (ibid.). One important aspect of this work is the existence of an explicit model of an enterprise which plays a role similar to the active glossary in the SBF approach. This aspect is not explicitly tackled in the VITAL-KR, which provides instead a more fine-grained, extendible, and customisable framework for integrating human and software agents.

**Software Engineering Approaches.** The issue of interoperability has also been extensively approached by the software engineering community. On one hand tools have been developed to integrate heterogeneous software components, but they provide a looser form of integration than non-embedded AI architectures. Typically there is no attempt at understanding the cooperation strategies which are needed, and at classifying the communication primitives. When this is attempted *ad hoc* protocols are defined, which are at a lower level than the ones used in AI. On the other hand tools based on specification languages provide a way of characterizing precisely the interaction between components, but, as is the case with embedded AI approaches, they are limited by the formalism and environment used, and by the master-slave cooperation framework.

6 Conclusions

The VITAL-KR has been implemented in Harlequin Common Lisp and runs on SUN workstations. The system has been tested on a number of applications and the results have been encouraging, both from a methodological and an efficiency related point of view.

In conclusion we believe that the VITAL-KR provides a number of advantages over alternative AI programming environments, which make it well-suited for the development of hybrid, knowledge-based applications, comprising a number of cooperating agents. Its communication primitives are *generic*, as they do not depend on the structure of a particular knowledge representation system; it is *extendible*, as it provides well-defined mechanisms for integrating new software modules; it enjoys a formal, unambiguous specification; it includes mechanisms to ensure the consistency of the overall, hybrid KB; and it provides a uniform mechanism for designing and integrating both software and human agents.

**Acknowledgements**

We would like to thank John Domingue and Stuart Watt for many useful discussions on a number of topics related to this paper.
Bibliography


[Len90] Lenat, D.B. and Guha, R.V. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Reading, MA.


