



The power of distributed software agents
by Lance Dean Kind

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Computer Science
Montana State University
© Copyright by Lance Dean Kind (1995)

Abstract:

As the Internet continues to grow, a more efficient method than those currently used is needed for knowledge sharing. This method must be flexible and powerful enough to solve the following problems: Heterogenous Communication, Peer to Peer Processing, Exchange of Information, and Information Routing.

Research has shown that distributed software agents are capable of surmounting these distributed computing problems. Agents are looked at in general with specific examples illustrated with KQML agents. Agents, RPCs, and distributed databases are compared to see how they solve or do not solve these problems. It was discovered that agents can solve these problems to an extent which RPCs and distributed databases cannot.

THE POWER OF DISTRIBUTED SOFTWARE AGENTS

by

Lance Dean Kind

**A thesis submitted in partial fulfillment
of the requirements for the degree**

of

Master of Science

in

Computer Science

**MONTANA STATE UNIVERSITY
Bozeman, Montana**

April 1995

©COPYRIGHT

by

Lance Dean Kind

1995

All Rights Reserved

N378
K5749

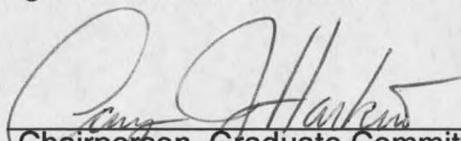
APPROVAL

of a thesis submitted by

Lance Dean Kind

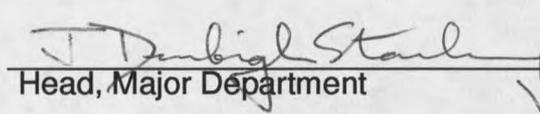
This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

4/26/95
Date


Chairperson, Graduate Committee

Approved for the Major Department

4/26/95
Date


Head, Major Department

Approved for the College of Graduate Studies

5/4/95
Date


Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Lance Kind
Date 4-26-95

TABLE OF CONTENTS

	Page
1. INTRODUCTION.....	1
What is an Agent?.....	2
2. HETEROGENOUS COMMUNICATION.....	4
Agent Communication.....	4
KQML.....	5
Other Methods of Distributed Computing.....	7
RPCs.....	7
Distributed Databases.....	8
3. PEER TO PEER PROCESSING.....	9
KQML Agents.....	12
4. EXCHANGE OF INFORMATION.....	14
KQML.....	14
Shop-Talk.....	16
Agentifying Current Software.....	18
KIF.....	20
Ontology.....	20
Other Methods of Distributed Computing.....	21
RPC.....	21
Distributed Databases.....	22
5. INFORMATION ROUTING.....	23
Agent Name Servers.....	23
Content Based Routing.....	25
Facilitators.....	25
Federation.....	26
RPC.....	27
6. SUMMARY.....	29
REFERENCES.....	30

LIST OF TABLES

Table	Page
1. KQML String Syntax in EBNF.....	5
2. Implemented KQML Performatives.....	6
3. Fred's Agent Requesting Information on Roofing Nails.....	12
4. Form of a KQML Performative.....	14
5. EBNF Form of Shop-Talk's Syntax.....	16
6. Pseudocode and EBNF of Shop-Talk Returned Values.....	17

LIST OF FIGURES

Figure	Page
1. Client Server Paradigm.....	9
2. Peer to Peer Process Communication.....	10
3. Transducer Agent.....	18
4. Wrapper Agent.....	19
5. ANS Hierarchy.....	24
6. Facilitators and Agents in a Federation.....	26
7. RPC Location Broker Routing Scheme.....	27
8. Agent n Uses Multiple Routing Schemes.....	28

ABSTRACT

As the Internet continues to grow, a more efficient method than those currently used is needed for knowledge sharing. This method must be flexible and powerful enough to solve the following problems:

Heterogenous Communication,
Peer to Peer Processing,
Exchange of Information, and
Information Routing.

Research has shown that distributed software agents are capable of surmounting these distributed computing problems. Agents are looked at in general with specific examples illustrated with KQML agents. Agents, RPCs, and distributed databases are compared to see how they solve or do not solve these problems. It was discovered that agents can solve these problems to an extent which RPCs and distributed databases cannot.

CHAPTER 1 INTRODUCTION

With the increased popularity of the Internet, organizations have been trying to find effective ways to communicate between different sites worldwide. At this time information is being disseminated through the network but not always in a user friendly manner; the user must often learn to use cryptic software like ftp and telnet. What is needed is some common method of sharing knowledge which can take advantage of network structures.. This common method must be flexible enough and robust enough to surmount the following problems:

- 1) Heterogenous Communication: This is due to the fact that there are a wide variety of platforms and operating systems in operation on the Internet.
- 2) Peer to Peer Processing: The client server paradigm is limiting in cases where asynchronous transactions are an advantage.
- 3) Exchange of Information: Flexibility is needed so that all types of information can be represented and shared.
- 4) Information Routing: Routing is an important and often overlooked problem; if one cannot find the pertinent information, then the information is as good as useless.

Research has shown that not only can distributed software agents solve these problems; but they often do it better than other methods of distributed computing such as RPCs and distributed databases.

What is an Agent?

Three definitions of an agent supplied by a wordprocessor (Excellence!) are:

- 1) one by which work is accomplished or an end effected;
- 2) one who acts for another; and
- 3) one who keeps secret watch to obtain information.

The first two definitions are the spirit of why one would want to have a software agent. The last one could be facilitated by a software agent, but is unethical.

M. Genesereth and S. Ketchpel define a software agent as:

- 4) "software 'components' that communicate with their peers by exchanging messages in an expressive agent communication language." [4]

This definition implies that not only should an agent be able to perform definitions one through three, an agent should communicate with other software agents. All four of these definitions imply a behavioral condition for agenthood; i.e. if a piece of software exchanges messages in an agent communication language then it is an agent.

Discussions about how agents can solve the four problems brought up in the introduction will be organized in the following manner: CHAPTER 2 HETEROGENOUS COMMUNICATION; CHAPTER 3 PEER TO PEER PROCESSING; CHAPTER 4 EXCHANGE OF INFORMATION; and CHAPTER 5 INFORMATION ROUTING.

CHAPTER 2 HETEROGENOUS COMMUNICATION

If effortless information sharing is to be achieved, heterogenous communication is a requirement. A solution should be able to support multiple protocols including: TCP/IP, SMTP, and HTTP since these are the most prolific protocols running on the Internet.

Agent Communication

Agents can use many different communication protocols to connect to the Internet. KAPI (KQML APplication Interface) was chosen for implementing KQML (Knowledge Query Manipulation Language) software agents, for this thesis work, due to the wealth of communication protocols it supports [15]. Being able to support many of these low level transport protocols is only part of the problem. In order for different agents to have the ability to talk to each other across different low level transport protocols, a common high level protocol is needed. This high level protocol is called the agent communication language (ACL). Different agents need to have a common ACL to allow interagent communication. KQML, one such ACL, has been used in a variety of projects funded by DARPA, the Air Force, and the Navy. (See Reference section). Because of KQML's flexibility and popularity, KQML agents were implemented for the research shown in this thesis.

KQML

KQML is a speech act based ACL; the KQML transport was built to model human speech [3]. KQML messages are performatives which contain keywords and their expressions [1, 2, 6, 11,13]. See Table 1. (A detailed discussion on

Table 1 KQML String Syntax in EBNF

```

<performative> ::= ( <performative_name> {<whitespace> :<keyword>
                    <whitespace> <expression>}* )
<performative_name> ::= <word>
<keyword> ::= <word>
<expression> ::= <word> | <quotation> | <string> |
                ( {<word> {<whitespace> <expression>}* } | <performative> )
<word> ::= <character><character>*

```

KQML syntax and semantics is given in [1])

The KQML specification states that it is not a requirement to implement all 41 of the performatives listed in order to be KQML compliant. In fact, implementors may create more performatives for their own purposes if they find the reserved KQML performatives lacking. The performatives mentioned in the specification are reserved; if used, they must follow the specification requirements in order to be KQML compliant. Each performative has keywords which have certain values stored in their expression field as defined in the KQML specification [1].

Experimental agent implementations for this thesis work consist of two agents: one agent acts as a server which performs lookups in a database of merchandise (a catalog) of some fictional business; the other agent acts as a

client which accepts input from a user who is interested in searching and pricing some item. Table 2 shows five KQML performatives which were implemented; S

Table 2 Implemented KQML Performatives

<performative_name> ::= ask-if ask-about tell deny error	
Performative Name	Meaning
ask-if	S wants to know if the sentence is in R's VKB
ask-about	S want all relevant sentences in R's VKB
tell	the sentence in S's VKB
deny	the embedded performative does not apply to S
error	S considers R's earlier message to be malformed.

is the sending agent and R is the receiving agent. As requests for item descriptions, and prices of items are sent to the merchandising agent (MAG), it performs operations upon its catalog (database) and returns a performative (tell, or deny) which reflects what the MAG has discovered in its database (DB) .

Other Methods of Distributed Computing

Agents compare favorably to other ways of creating distributed applications. The pros and cons are discussed below.

RPCs

RPCs support only one transport mechanism: TCP/IP. This is only a slight disadvantage since most machines, which are capable of network access, have TCP/IP. The primary disadvantage is that different vendors have their own systems of implementing RPCs. If an application is written using Sun RPCs, a DCE RPC application will not be able to communicate with it. This is because RPCs operate on a lower level than agents; binary data is marshaled into a form for transport, and then delivered to the network, and then to the remote machine. The marshaled form (called an interface definition language or IDL) is different between RPC standards.

RPCs are implementation dependent while KQML does not try to force a specific implementation. KQML is concerned with interagent communication; not with data representation or transport. However, since RPCs are closer to the hardware and operate on a level closer to the transport protocol, they typically have higher performance for applications where computational speed is important.

Unlike RPC's, agents are often machine and operating system independent. Additionally, KQML is ASCII based so architectural characteristics (like big endian versus little endian) are not significant and no data marshalling is needed. The drawback to ASCII representation is that data is likely to require more space and network bandwidth than a binary.

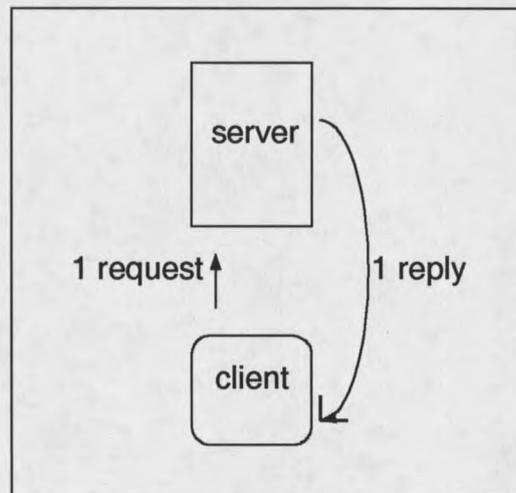
Distributed Databases

Distributed databases (DDB) allow data to be searched and retrieved from remote sites. Portions of the DB may be physically stored at different sites to allow load sharing. Some DDB use RPCs to perform operations on data. However much like RPCs, each DDB implementation only communicates with DDB of the same implementation. Different DDBs may be able to share data after some type of conversion process is applied.

CHAPTER 3 PEER TO PEER PROCESSING

The client server model is the predominate model of software interaction on the Internet. Figure 1 illustrates the client server paradigm. The client makes

Figure 1 Client Server Paradigm

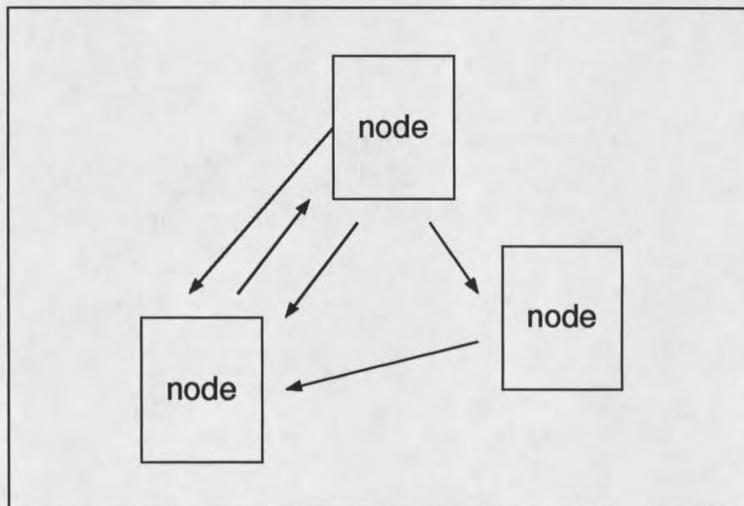


a request and the server responds by sending a reply in fulfillment of the request. The communication is synchronous in nature since the communication takes place in a lockstep fashion. The server never acts independently from a client. This model is too restrictive if the server must be proactive. For example: If the

server in Figure 1 was a MAG, the owner could benefit if it could actively search out new clients. If peer to peer processing were allowed, the MAG could communicate with other agents which represent their owner's interests. The MAG could leave its address and a message as a "calling card". Later if the owner of the agent is looking for an item, the agent could reference its DB of agent addresses and interests; and contact the MAG for further information.

In peer to peer processing, one node may actively seek out a service like a client, and later it may serve a request like a server. Figure 2 illustrates this concept. With asynchronous communication, the replies are received in a nondeterministic fashion. Each node does not know when a reply or a request

Figure 2 Peer to Peer Process Communication



will arrive. The extra work involved in peer to peer as opposed to client server is that the nodes have to keep track what reply went with which request. This extra bookkeeping isn't needed in client server since transactions occur in lockstep.

When an agent receives a message and decides that it is valid, it executes the command or program which is embedded in the message. In this fashion, a client agent's program or command is executed in the server agent's address space. If there are return values, they are returned back to the client agent.

KQML Agents

The KQML ACL supports peer to peer bookkeeping through keywords which are imbedded in the performatives. Table 3 shows an example of an agent's request and the response received from the serving agent. This example

Table 3 Fred's Agent Requesting Information on Roofing Nails

Fred's Agent	
Request:	Response Received:
(ask-if	(tell
:receiver joes_ hardware	:receiver fred
:sender fred	:sender joes_ hardware
:language shop-talk	:language shop-talk
:ontology shop-talk	:ontology shop-talk
:reply-with (nails 3-35-95, 3:00 PM)	:in-reply-to (nails 3-35-95, 3:00 PM)
:content (find (roofing nails))	:content ({{(124,roofing nails)}})
))

was generated with a MAG and a shopping agent; the scenario is as follows:

Fred is looking for roofing nails. He starts up his agent and asks it to search Joes Hardware to see if it carries roofing nails. Notice the keyword reply-with and its expression in Fred's request. In the tell performative which Fred's agent received as a server, the in-reply-to keyword has the same expression as the reply-with expression. This allows the agent to collate the messages it receives by context; which is defined by the reply-with, in-reply-to, and sender expressions. Notice that the date and the time are included in the context. This is so that if multiple queries are made to Joes Hardware's agent, the response's true context will be known. One more improvement that could be made is to add a random number into the :reply-with <expression>. This would allow the context to be understood unambiguously in case Fred's agent can send out more than one request per second. Since Joes Hardware's agent did not put a reply-with keyword in its performative, it is not expecting a reply.

Agents collecting information for their owners may send out dozens of requests to other agents. Instead of waiting for each response as in the client server situation, they may continue with other tasks. While doing other tasks, the agents may receive responses concurrently. The agents will organize the information they receive by context in their DB and output reports to the owners as they make progress.

CHAPTER 4 EXCHANGE OF INFORMATION

The ability to exchange any information (that can be represented in a computer) is the ability to represent all types of information: binary, ASCII, extended ASCII, and knowledge itself.

KQML

KQML speaking agents have the ability to exchange all forms of data because KQML does not care what is stored in the content expression. See Table 4. What is placed in the content expression is up to the implementor; it could be binary, ASCII, source code of a program,.. etc. KQML is used by agents

Table 4 Form of a KQML Performative

```
(<performative_name>  
:receiver <expression>  
:sender <expression>  
:language <expression>  
:ontology <expression>  
:reply-with <expression>  
:content <expression>  
.....  
)
```

to talk to each other about data stored in the content expression.

The receiving agent discovers what is in the content expression by looking at the language field. This field describes the language of the content field. Notice that even though a given set of agents may be able to communicate due to a common ACL, they may not be able to understand the data in the content expression. They can communicate with each other about the data but they need not understand the data.

For example, if agent 99 wants to exchange a lisp program with agent 007 (a prolog executing agent), 007 is going to get the lisp program and look at the language field and see that it is not prolog. If 007 is a sophisticated agent, then it could recruit the help of another agent to run the lisp program and then receive the output from the lisp running agent. If 007 isn't smart enough to do this or 007 can't find a lisp executing agent, then 007 will have to respond with an error performative; its comment expression could contain a plea of ignorance about the sender's language.

Shop-Talk

The MAG agent and the potential client agent must speak a language/protocol called shop-talk. Table 5 describes the shop-talk language in

Table 5 EBNF Form of Shop-Talk's Syntax

```

<shop-talk function> ::= (<command>)
<command> ::= {<description command> (<string>)} |
               {<catalog command> (<catalog number>)}
<description command> ::= find
<catalog command> ::= price | describe
<string> ::= {<upper or lower alpha char> | <number> |
               <all punctuation except: \, (, ), {, }, and ", ">}+
<catalog number> ::= <all integers>

```

EBNF. The shop-talk language expresses queries to a MAG about its catalog. One could use the find command to search an online catalog for some item which is described in the <string> field. Once the catalog number was found, the price or the entire description of the item could be requested. The <string> definition

was restricted to make parsing the content expression easy.

Table 6 shows the form of the returned data; the MAG replies to shop-talk

Table 6 Pseudocode and EBNF of Shop-Talk Returned Values

```

<shop talk fct output> ::= "{" <ordered pair> {,<ordered pair>}* "}"
<ordered pair> ::= (<catalog number>,<data>)
if <shop talk fct output> is find | describe
    <data> ::= <description>
else
    <data> ::= <real number>

```

functions by returning a set. The set contains ordered pairs; the left part of the pair is always a catalog number so that describe and price would be one to one functions. The number of ordered pairs returned is determined by the context of the KQML performative: if the performative_name was ask-if, then the first match would return; or if the performative_name was ask-about, then all the matches which exist in the receiver's DB would return. Other KQML speaking agents can talk to a MAG agent but only those that understand shop-talk can query the catalog.

Agentifying Current Software

The value of having distributed access to data is obvious, but it is unlikely that people will want to rewrite the software they are using. However, since agenthood is defined by how a software component acts, software need only to speak in an ACL to become an agent. The following paragraphs discuss some techniques which allow legacy software to be made into agents.

Software for which there isn't any access to the source code can be agentified by writing a transducer agent [3]. The transducer agent accepts messages from other agents and translates the messages into a form which is acceptable to the software and then executes the software on the input. See Figure 3. The agent then captures the software's response and packages it into

Figure 3 Transducer Agent

