



UNIVERSITY OF
CALGARY

A TUTORIAL ON AGENT COMMUNICATION AND
KNOWLEDGE SHARING

PRESENTED

BY

Cyprian Foinjong Ngolah

SENG609.22
AGENT-BASED SOFTWARE ENGINEERING

Course Instructor: Dr. Behrouz H. Far

AGENT COMMUNICATION AND KNOWLEDGE SHARING

Abstract

With the rapid technological development particularly in the computer industry, many computer programming languages are being developed with many computer programs written in these languages. Writing good programs takes time and effort. It is therefore necessary that programs once written should be able to work together (interoperate) with each other so that a program written in one language can benefit from the advantages that may be associated with another program written in another language. One of the problems with interoperation of programs is the heterogeneity of the programs. This is because programs are written by different people, at different times, in different languages and with different interfaces. Agent-base Software Engineering is an attempt to facilitate the creation of software products that can easily interoperate in such settings.

In Agent-base Software Engineering, programs are written as software agents (software components) that communicate with each other by exchanging messages through a communication language. For a successful communication in such heterogeneous environments, the agents have to share knowledge with each other.

The aim of this tutorial is to discuss agent communication and knowledge sharing. A discussion about agent communication, methods of agent communication, agent communication languages, content languages and knowledge sharing between agents is presented together with constraints that may inhibit successful knowledge sharing and agent communication.

1 Introduction

Many software products available to users today provide a wide variety of services in various domains. Most of these programs provide significant values to the users when used in isolation. However, there is a significant demand for computer programs to interoperate (i.e. exchange information and services) with other programs. Such interoperation will enable them to solve problems that cannot be solved by individual programs. Moreover, software development is expensive and complex. It is therefore beneficial for software products to be shared and reused as this will drastically reduce the effort and money spent in building software each time from scratch.

One of the problems with interoperation of programs is the heterogeneity of the programs. This is because programs are written by different people, at different times, in different languages and with different interfaces. Agent-base Software Engineering is an attempt to facilitate the creation of software products that can easily interoperate in such settings. In Agent-base Software Engineering, programs are written as software agents (software components) that communicate with each other by exchanging messages through a communication language. For a successful communication in such heterogeneous environments, the agents have to share knowledge with each other.

The aim of this tutorial is to discuss agent communication and knowledge sharing between agents. Section two discusses agent communication, agent communication languages, data representation languages, and communication architectures. Section three discusses knowledge sharing in agent-based systems, knowledge representation and constraints on knowledge sharing. Section four is the conclusion of this tutorial while sections five and six are acknowledgement and references to related works respectively.

2 Agent Communications

Agent Communication can be defined as the exchange of information between software agents. Tim Finin defines a software agent as an autonomous, goal directed process that is situated in, is aware of, reacts to its environment and cooperates with other agents to accomplish tasks [Finin, 1999]. Similarly Genesereth defines a software agent as any system that uses an Agent Communication Language(ACL) to exchange information [Genesereth, 1992]. From these two definitions, we see that for an agent to be able to cooperate with other agents and react to its environment, it needs some agent communication language. Communication between agents encourages autonomy and discourages any regard for other agent's internal structure. It encourages the existence of societies of agents that are able to provide solutions to more complex problems. In particular, communication between agents makes it possible for programs written by different programmers, in different languages, using different interfaces to communicate. For this to be possible, agents need a common communication language that is more concerned with the information exchange than its content. Communication is generally broken down into three sub-sections. These are:

- 1) **Interaction protocol:** This refers to the high level strategy pursued by the software agents that governs its interactions with other agents. Such protocols include negotiation schemes, game theory, auctions, contracts, etc.
- 2) **Communication Language.** This is the medium through which the attitudes regarding the content of the exchange message are communicated.
- 3) **Transport protocol.** This is the actual transportation mechanism used for communication using the communication language. Such protocols include TCP, IP, SMTP, HTTP, IIOP, etc.

2.1 Methods of Communication

Various agent communication methods are in used. Some communication architectures commonly used include the blackboard communication architecture, message architecture and facilitation architecture. In the blackboard communication architecture, information is made available to all agents in the system through a common information space and there is no direct communication [Figure 1A]. In the Message architecture, there is a direct exchange of messages between agents using a common language in a conversational style [Figure 1B]. In this case, the sending agent specifies for whom the message is intended and the receiving agent accepts the message when it reaches it. Two very popular approaches of direct communication include contract-net and specification sharing. In the contract-net approach to interoperation [Davis et al, 1983], agents in need of services distribute their requests for proposals to other agents and the recipients of

these messages evaluate those requests and submit bids to the originating agents to decide which agents to award the contract to. In the specification sharing approach to interoperation, agents supply other agents with information about their capabilities and needs and these agents can use this information to coordinate their activities and respond accordingly.

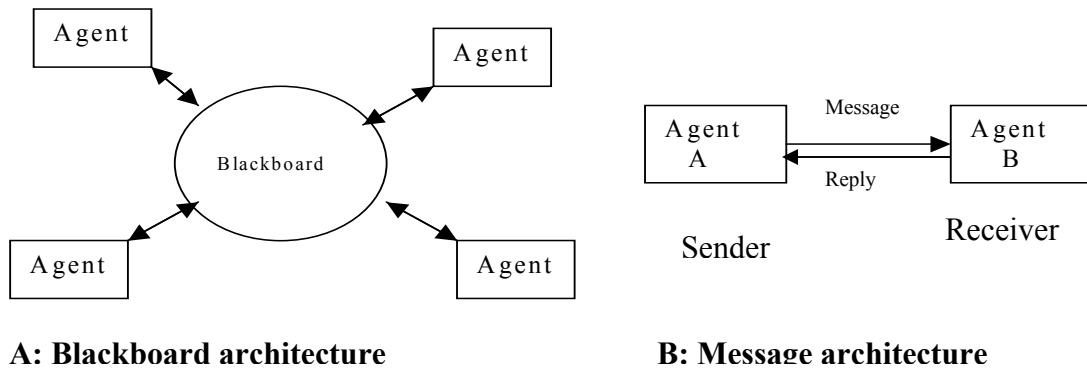


Figure 1 : Methods of Communication

Direct communication however, has some disadvantages. One of the problems with this form of communication is cost. If the agent community is large like the Internet, then the overheads in broadcasting bids and processing requests are quite high. Another problem with this is the implementation complexity.

A popular way of enhancing communication between agents is to implement the system through facilitation. Here agents are organized into a federated system where agents do not communicate directly with each other but through system programs called facilitators. In a federated system, agents use an agent communication language to document their needs and capabilities for their local facilitators. Facilitators use the documentation provided by these agents to transform application-level messages and send them to appropriate places. The agents in a federated system surrender their autonomy to the facilitators and facilitators take full responsibility for their needs. Figure 2 shows a simple federated architecture of a multi-agent system in which there are multi-agent systems with each system of agents controlled by a facilitator. The facilitators communicate among themselves to express the needs and desires of their respective agents.

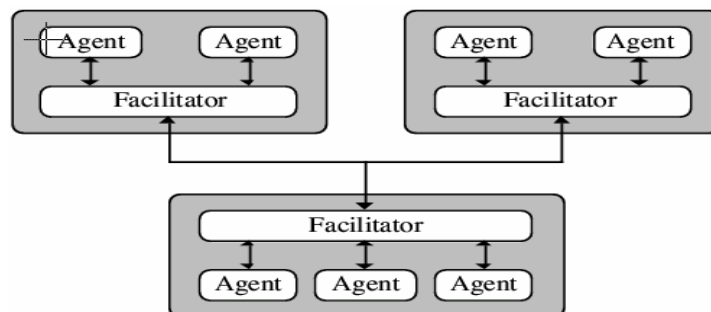


Figure 2: Communication through a Federated system of agents.

2.1.1 Communication Strategies

Common communication strategies include negotiation, auctions and contracts. The concept of contracts has already been briefly explained in section 2.1. In negotiation, agents reach an agreement on joined actions in order to achieve a common or separate goal. In auctions, there is a centralized redistribution of resources in the system. Communication support by services and agents may include authentication, registration, matchmaking and brokers/facilitators.

2.2 Agent Communication Languages (ACL)

An Agent Communication Language (ACL) is one in which agents can communicate with other agents in order to accomplish tasks. Such a language should be one where inconsistencies and arbitrary notational variations are eliminated. In search for such a language, researchers in ARPA Knowledge Sharing Effort [Neches, 1991] defined the components of an ACL that is able to meet such standards. The ACL has three main parts – a vocabulary, an “inner” language called KIF and an “outer” language called KQML. According to this definition, an ACL message is simply a KQML expression in which arguments are sentences in KIF formed from words in the ACL vocabulary. Typically, an ACL is structured in three layers 1) a common agent communication protocol, 2) a common format for content of communication message and 3) a shared ontology (semantics). Agent architectures therefore need both a representation and communication models where representation models include ontologies that define the domain model or vocabulary of the domain of discourse and content languages that represent the agent’s mental model of the world (e.g. beliefs, desires and intentions). ACLs are sometimes referred to as wrappers languages because they implement a knowledge level communication protocol that is not aware of the choice of the content language and the ontology specification mechanisms. The following subsections are brief discussions of the different layers of an ACL.

2.2.1 Common Agent Communication Languages

Various Agent Communication message-oriented languages are in used. Some of these include KQML, FIPA-ACL and KQML Lite. A brief introduction to each of these languages is given below.

2.2.1.1 Knowledge Query and Manipulation Language (KQML)

This is a high level interaction message-oriented, communication language and protocol for information exchange independent of content, syntax and ontology [Finin, 1999]. KQML is independent of any low-level transport mechanisms (e.g. TCP/IP, SMTP, HTTP, email, CORBA, etc) and high-level protocols such as **contract net** and **auctions**. Its architecture supports a community of agents with each owning and managing a Virtual Knowledge Base (VKB) that represents the mental model of the world. Agents can query and manipulate the contents of each other’s VKB using KQML. The KQML

specification defines the syntax for a collection of messages called performatives, which define the language in which the agents communicate. KQML has a reserved set of performatives that are commonly used in agent-agent or agent-community communication. Figure 3 shows KQML performatives developed in 1997 [Labrou, Finin,1997].

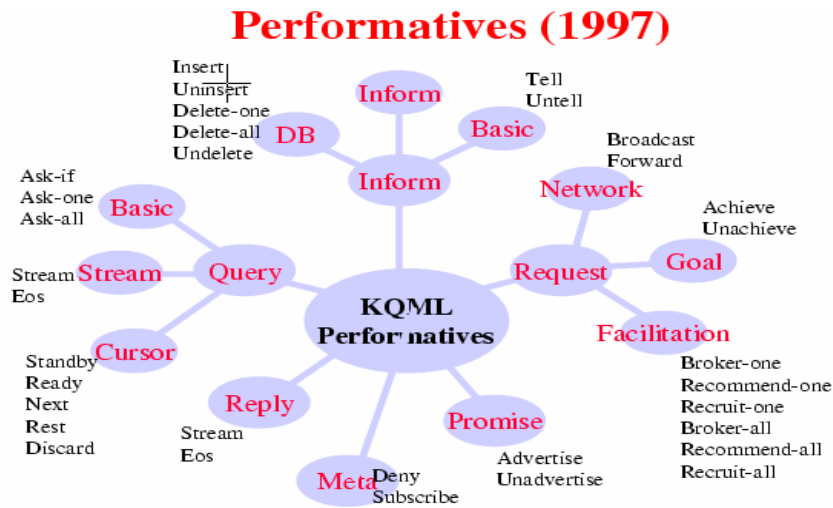


Figure 3: KQML Performatives [Finin,1997]

The use of each performative depends on whether it is applied to agent-pairs or to larger communities of agents and on the Knowledge Base Management System (KBMS) level communication function. Table 1 below shows sample performatives and when they can be applied.

Function Class	Performative	Level of application
Query and Response	ask-if, ask-all, ask-about, ask-one, tell, untell, deny, sorry	Agent-pair
Advertise or commit to a capability	advertise, unadvertised	Agent-community
Communication primitives other than pure asynchronous messages	Broadcast, forward, standby, subscribe	Either

A typical KQML message could be

```

performative → (tell : sender      Agent A
                  receiver        Agent B
                  in-reply-to     id7.24.02.5678
parameter → ontology            book14
                  language       C++
value → content → "price(ISBN429459,$30)")
  
```

In this message, Agent A is replying message “id7.24.5678” to Agent B about the price of a book “book14” where the original message from Agent B is implemented in a language called C++. KQML does not need to know the content of the message or the language in which it is written but transmits the message to the receiving agent and carries the reply back to the sender.

KQML, examples of communication

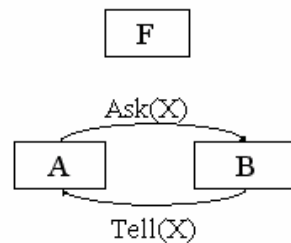


Figure 4: Direct Communication

A is aware of B, point-to-point protocol

In this communication, agent A wants to find out about X and knows that B has X in its knowledge base. So it sends a message directly to agent B and agent B replies by sending an answer to A directly.

KQML, examples of communication

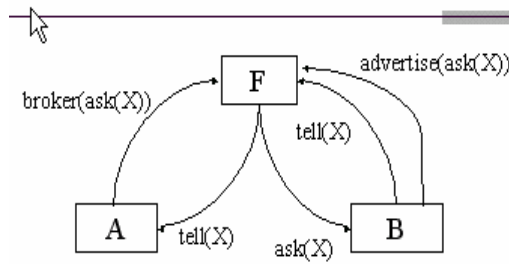


Figure 5: Indirect Communication

Broker performative, ask a facilitator agent to find another agent

KQML also provides a basic architecture for knowledge sharing through a special class of agents called **communication facilitators**, which coordinate the interactions of other agents. The example in figure 5 shows a simple example of this type of architecture. Here agent A wants to execute **ask(X)** but does not know which agent can process it. It then places the responsibility with the facilitator F to find an agent to process **ask(X)**. F finds that B is willing to process **ask(X)** and sends the query to B, gets the response and sends it to A.

2.2.1.2 FIPA-ACL

This is an agent communication language associated with FIPA’s agent open architecture. It is similar to KQML in that it works with any content language and ontology specification. However, there are many fundamental differences between FIPA-ACL and

KQML. For example, FIPA –ACL semantic models does not allow an agent to directly manipulate another agent’s Virtual Knowledge Base. Consequently, keyboard-editing performatives found in KQML (i.e. insert, delete, undelete, etc) cannot be used in FIPA-ACL. FIPA-ACL does not provide any performatives for cursor manipulation but KQML does. The FIPA architecture has an Agent Management System (AMS) specification, which specifies services that manage agent communities. As a result, FIPA-ACL does not have performatives that explicitly manage agent communities as is the case with KQML.

2.2.1.3 KQML Lite

This is an attempt to merge FIPA-ACL and KQML and is intended to provide a more powerful communication language than either KQML or FIPA-ACL[Venu, 1998]. It has more performatives than any of the two. However, this is not widely used in industry.

2.2.2 Content Languages

Content languages enable agents to reason and represent information about other agents. Such information includes information about models of other agents, beliefs, desires, intentions, perceptions and plans of the other agents. Content languages enable the agents to communicate the same range of knowledge.

Various content languages are in used with ACL. Some of these include KIF, SL (Semantic Language), Loom, Prolog, CLIPS, SQL, FIPA-SL, etc. One common characteristic of these languages is their expressiveness and neutrality. They should be interlingua for a wide range of systems. In this case a content language should be able to support communication from different languages. The most widely used content language is the Knowledge Interchange Format (KIF).

2.2.2.1 Knowledge Interchange Format (KIF)

KIF is a prefix version of first order predicate calculus with various extensions to enhance its expressiveness [Genesereth,1992]. It can be described as an interlingua language for encoded declarative knowledge. Figure 6 shows KIF as an interlingua between a knowledge base in KIF and three other languages. It is a language for the syntactic aspects of representation for knowledge sharing.

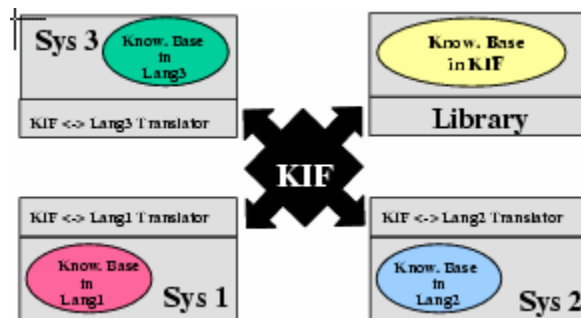


Figure 6: KIF as an interlingua between different representation languages

KIF provides for the expression of simple data and has implementation independent semantics. It can be used to express information in a simple database or to express more complicated pieces of information as the need arises. For example, the following three lines represent three statements in KIF.

```
(studentID John 234542)
(studentID Peter 433212)
(studentID Mary 212122)
```

These three lines encode three tuples in a student database where the first line expresses the fact that John is a student with a student ID 234542, the second line says that Peter is a student with student ID 433212 and son on. KIF can also be used to express complicated expressions. For example $(\Rightarrow(\text{and}(\text{real-number } ?x)(\text{even-number } ?y))(\text{>}(\text{expt } ?x?y) 0))$ is a KIF expression which says that if x is a real number and y an even number then the $x^{\text{expt}(y)}$ is greater than or equal to 0.

KIF is able to encode knowledge about knowledge using logical operators (negation, conjunction, disjunction, rules, quantification, etc). Despite the good features in KIF, there are some disadvantages. For example, it is quite complicated building fully conforming systems using KIF.

KIF Software

The IBM Agent Building Environment and RAISE reasoning engines use KIF as their external language. There exist parsers for KIF, which take KIF strings into C++ and Java objects. The Stanford Ontolingua, a language for building, publishing and sharing ontologies, uses KIF as its internal language.

3 Knowledge Sharing

In conventional Software Engineering paradigms, individual programs may exchange information among themselves. However, they lack knowledge at individual program level. Consequently there is little knowledge sharing between them. Knowledge sharing is defined as the exchange of information between two software systems or the ability for two software agents to reason and learn about each other's intentions, desires or beliefs. To be able to share knowledge the two agents must each have knowledge in their respective knowledge bases. Building an agent's knowledge base is expensive and difficult. It is therefore necessary that agents should share knowledge so that with the shared knowledge they can be able to accomplish tasks that individual agents could not have accomplished. To create knowledge bases, there must be a representation format that is used to represent the knowledge in the knowledge bases. This calls for a common knowledge representation language and format. However, sharing and reusing knowledge has some associated constraints.

3.1 Constraints on Agent Knowledge Sharing

1) Heterogeneous Representation Languages: There is no single knowledge representation that is best for all problems. As a result, sharing and reusing knowledge will involve translating from one representation to another. Tools are therefore needed that can help automate the translation process.

2) Heterogeneous Ontologies: Often the vocabulary used in each representation language is different. Consequently it is difficult to establish a good communication between two knowledge bases that are built using different representation languages. There is therefore, the need for a shared ontology if agents must share knowledge. However, for the shared ontologies to be useful, the definitions provided must include declarative constraints that specify the semantics of the terms being defined, and the ontology must provide procedural methods that enforce those constraints when the terms are used in an application.

To provide a solution to the first constraint, KIF is used as a representation language that is able to provide a common platform on which knowledge is represented. To provide a solution to the second constraint, an ontology development environment called **Ontolingua** has been developed as a language for building, publishing and sharing ontologies. Ontology is defined as the “set of concepts or terms that can be used to describe some area of knowledge or build a representation of it [Far, lecture notes, 2002]. Tim Finin defines ontology as a common vocabulary and agreed upon meaning to describe a subject domain [Finin, 1999]. From these definitions, we see that knowledge sharing in agents requires a communication language as well as shared ontology. Shared ontology can be explicit or implicit. Implicit ontology can be represented only by procedures whereas explicit ontologies are given a declarative representation in a well-defined knowledge representation language. There are three important aspects of explicit ontologies. These are:

Conceptualization: This involves the underlying model of the domain in terms of objects, attributes and relations.

Vocabulary: This involves assigning symbols or terms to refer to those objects, attributes and relations.

Axiomization: This involves encoding rules and constraints, which capture significant aspects of the domain model.

A Knowledge Representation Language suitable for knowledge representation should provide two important concepts.

- 1) A logical *Formalism* that includes a syntax for well-formed formulae, a vocabulary for logical symbols and an interpretation for logical symbols.
- 2) An *Ontology* that provides a vocabulary for non-logical symbols, definition of symbols and axioms constraining the interpretation of primitive symbols.

Thus a Knowledge Representation Language (KRL) = Logic + Ontology.

With a sound representation of knowledge at the agent's individual level, knowledge sharing is achieved through the interoperation of programs. One way of enforcing knowledge sharing is through facilitation. Agents use the ACL to supply machine-processable information to the facilitators, which coordinate the activities of the agents. Each agent registers and specifies its interest and capabilities to the facilitator. The facilitator then 1) forwards requests to an agent that can handle it, 2) decomposes a request if different parts of the request are to be handled by different agents and combines the different pieces of the request to generate an answer, 3) forwards requests to agents that have expressed interest, and 4) translates information between vocabularies to match an agent interface. The facilitator's task is facilitated by the ACL layered structure where KQML is responsible for transferring information from one agent to another, KIF acting as an interlingua between the different representational languages that might have been used in the different knowledge bases, and the shared ontology responsible for enforcing a well defined knowledge representation. In agent-based technology, knowledge sharing is an integral part of the communication mechanism. This can be achieved through interaction, cooperation, coordination and competition.

4 Conclusion

In this tutorial, a discussion about agent communication and knowledge sharing has been presented. Agent communication has been defined as the exchange of information between agents while knowledge sharing is the phenomenon where agents are able to exchange information among themselves as well as reason and learn about other agents.

For agents to share knowledge, they need to communicate using a common language called an Agent Communication Language (ACL) which can be divided into syntax, semantics and pragmatics. The syntax of the language is described using a content language that enables agents to reason and represent information about other agents such as models of other agents, beliefs, desires, intentions, etc. The semantics are described in the ontology, which defines a common vocabulary and describes the domain, representation of common source code elements, system specific details and knowledge sharing axioms (e.g. wants-to-know, need-to, etc). The pragmatics are based on KQML, which describes the way agents communicate using performatives.

Acknowledgement

I greatly acknowledge the contributions from the authors whose works are cited in this document. The work could not have been completed without their varied and useful ideas.

References

[Finin,1993] Finin, T., Y. Labrou, Agent Communication Languages, “Agent Communication Languages Tutorial, University of Maryland Baltimore County, October, 1999.

[Genesereth,1992] Genesereth, M.R., Fikes, R.E. et al. Knowledge Interchange Format Version 3 Reference Manual, Logic-92-1, Stanford University Logic Group, 1992.

[Venu,1998] Venu V. “Comparing Agent Communication Languages”, OBJS Technical Note, July 1998.

[Labrou, Finin, 1997] Labrou, Y. T., Finin, A Proposal for a new KQML Specification, TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore County, <http://www.cs.umbc.edu/kqml/papers/kqml97.pdf>

[Michael, Genesereth] Michael, R. Genesereth, Steven, P. Ketchpel “Software Agents Stanford University Logic Group,
<http://logic.stanford.edu/sharing/knowledge.html>.

[Finin,Labrou, Mayfield, 1997] Finin, Labrou, Mayfield, “KQML as an agent communication language” in Bradshaw, J., “Software agents”, The MIT Press, 1997

[Dignum, 2000] Dignum, F., Greaves, “Issues in agent communication: an introduction”, Springer Verlag, 2000

[Far, 2000] Far, B.H, Seng609.22 Lecture Notes: Agent Based Software Engineering, Fall, 2002.

[Sanjeev, 2000] Sanjeev, K., M.J. Huber, D.R. McGee, P.R. Cohen, and H.J. Levesque: “Semantics of Agent Communication Languages for group Interactions”,
<http://www.cse.ogi.edu/CHCC/Papers/sanjeevPaper/aaai2000.pdf>

[Davis, 1993] Davis, R., and R.G. Smith: “Negotiation as a Metaphor for Distributed problem Solving I Artificial Intelligence”, 06, 1(1993), 63-109..

[Neches, 1991] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R. Swartout “Enabling Technology For Knowledge Sharing”, *AI Magazine, Volume 12, No. 3, Fall 1991*