Distributed Artificial Intelligence and Multi-Agents

Dr. Rajveer S Shekhawat

With the empowerment of computing with the high-speed networks, AI has found a new incarnation namely Distributed AI, which can now solve some the real life problems using distributed versions of the AI algorithms. A new orientation to the application of AI has emerged termed as agents, which can reason on their own given the goals and the capability to glean the environmental information. With the replacement of wired communication by wireless protocols, the agents can be made more versatile in terms of mobility and constraint free communication. This has enabled DAI to be extended to colonies of agents opening a whole new frontier of research for imitating social behaviour of the intelligent agent societies which can co-operate and/or compete to achieve individual or group objectives. This paper introduces the concept of DAI and the enabling technologies for its implementation followed by the application to multi-agents. The different issues involved in each of the theme are briefly discussed.

Dr. Rajveer S. Shekhawat is Senior Scientist, Central Electronics Engineering Research Institute, Pilani, Rajasthan

Artificial intelligence is not a new field in computer science. It has existed since the man started thinking. The early AI researchers were philosophers and psychologists who wondered about the reasoning capability of human beings. But the AI research per se can be said to have started in early fifties when computers became available to implement the little understanding we had of human beings as artificial reasoning. The famous experiment of Alan Turing was the first attempt to formalize the AI. Since then, our understanding of intelligence and capability its imitation has been on the rise. This has of course been made possible by the power of modern computers. One can see from these developments that the increasing intelligence of artificial machines/computers has been synonymous with the computing speed and increasing memory capacity of these. We have seen the first practical AI systems emerge starting with the mathematical theorem proving programs, game playing software and the most successful expert systems. The later are rule based reasoning systems, which can do inductive or deductive reasoning. The rules however always derived from human expert somehow. Fuzzy reasoning has been a feather in the cap of computing paradigms, which are very near in human reasoning mechanism. A host of new methodologies have since emerged variously called artificial neural networks, evolutionary algorithms, bio-inspired computing and so on.

The above developments however remained confined to only single AI entities. Whereas we know that human beings do not always think and work alone rather they delve in societies. Thus their thinking and behaviour is influence in a large manner by the neighbouring human beings alternately called family and social structure. As if we have reached the peak in imitating intelligence of human beings, a major segment of AI researchers has now focussed on how to imitate the human in the social framework. Psychologists and sociologists and linguists have built up a large amount of theory as the field is multi-disciplinary. Now computer scientists have joined this group and started implementing multiple intelligent entities, which think independently, communicate, negotiate

and take decisions affecting the group as a whole. This has been made possible by developments in the field of computer communications, networking and wireless connectivity. The networked communities have necessitated the emergence of intelligent entities also called agents. Intelligent agents implemented as programs are termed as software agents. If robots are embedded with intelligence and are able to co-operate for a given task, these are termed as multi-agents or humanoids. In this paper, an attempt has been made to put in the perspective the enabling technologies for multi-agents and swarm intelligence. In **section2,** how computers at remote locations can communicate with each other and understand and co-operate has been explored. The development of technologies such as components (DCOM, CORBA) etc has been included. Section three delves into how the distributed computing has been used to implement DAI. **Section 4** takes up issues involved in implementing software agents and multi-agents as communicating intelligent agents. Narrating the near future scenario concludes the paper.

2. Distributed Computing and Systems:

Distributed systems require that computations running in different address spaces, potentially on different hosts, be able to communicate. For a basic communication mechanism, the Java™ programming language supports sockets, which are flexible and sufficient for general communication. However, sockets require the client and server to engage in applications-level protocols to encode and decode messages for exchange, and the design of such protocols is cumbersome and can be error-prone.

An alternative to sockets is Remote Procedure Call (RPC), which abstracts the communication interface to the level of a procedure call. Instead of working directly with sockets, the programmer has the illusion of calling a local procedure, when in 3 fact the arguments of the call are packaged up and shipped off to the remote target of the call. RPC, however, does not translate well into distributed object systems, where communication between program-level *objects* residing in different address spaces is needed. In order to match the semantics of object invocation, distributed object systems require *remote method invocation* or RMI. In such systems, a local surrogate (stub) object manages the invocation on a remote object. However application of RMI is limited to JAVA only. DCOM provides a higher-level abstraction for developing distributed software. DCOM and CORBA are based on an object–oriented model. DCOM is a desktop-centric middleware developed by Microsoft, where as CORBA is an enterprise-focused middleware standard maintained by OMG.

A distributed system is collection of autonomous computers linked by a network and equipped with distributed system software. The distributed system software enables the comprising computers to co-ordinate their activities and share their resources.

Middleware: Middleware in distributed systems is of type of distributed system software, which connects different kinds of applications and provides distribution transparency to its connected applications .It is used to bridge heterogeneities that occurred in the systems. Middleware replaces session, presentation and application layers of OSI model. Figure one shows middleware layers as implemented in RMI.

Based on significant standards, middleware can be divided into several categories such as SOCKET, RPC, RMI, DCOM and CORBA. We shall briefly describe each of these.

2.1 Sockets: A Socket is a peer-to-peer communication endpoint. Sockets are generic interfaces, which enable processes that reside in different computers to communicate to each other.

Two processes communicate with each other via their sockets. The process that send data to another process is called sending process while the process that receive the data is called receiving process (Refer figure two).

2.2 Remote Procedure Calls (RPC): Software developed using RPC is easier to be ported compared to sockets. The strength of RPC lies in its ease of use, portability and robustness. Its ease of use is the result of the higher-level abstraction that to provides to the developers and RPC's similarity with normal procedure calls. RPC

mechanism, an extension of the procedure call mechanism in the sense that it enables a call to be made to a procedure that do not reside in address space of calling process. Both normal and remote procedure calls are usually synchronous and follow the request-reply mechanism, in which a client is blocked until its server responds to the call. A major issue in the design of an RPC facility is its transparency.

2.3 Remote Invocation Method (RMI): RMI gives a better transparency of low level details compared to RPC and Sockets. RMI is a Java based middleware that allows methods of java objects located in java Virtual machine to be invoked from another JVM even when this JVM is across a network. *Remote method invocation* (RMI) is the action of invoking a method of a remote interface on a remote object. Most importantly, a method invocation on a remote object has the same syntax as a method invocation on a local object.

RMI applications are often comprised of two separate programs: a server and a client. A typical server application creates a number of remote objects, makes references to those remote objects accessible, and waits for clients to invoke methods on those remote objects. A typical client application gets a remote reference to one or more remote objects in the server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application. A *remote object* is one whose methods can be invoked from another Java virtual machine, potentially on a different host. (Refer figure three)

2.4 Component Object Model (COM): Distributed COM (DCOM) is a middleware developed by Microsoft that provides a distributed framework based on object-oriented model. It is the distributed extension to the component object model, which provides an object remote procedure call. (ORPC) on top of RPC.DCOM services are divided in two parts : OLE and COM.COM provides the underlying object systems on which OLE services rely upon. COM services are

- uniform data transfer,
- monikers,
- persistence storage.

Uniform data transfer service is a COM service that provides the basic facilities to exchange data between applications .It extends the windows clipboard to handle OLE objects.

Monikers service provides the mechanism to name objects for their identifications.

Persistence storage device allows objects to be stored in the disk.OLE parts contain three services

- compound documents
- drag-and drop
- automation.

Compound documents services provides ability to link information in a document through three services: in-place activation, linking and embedding. In-place activation enables container applications to hold component objects, permitting the user to manipulate component application operations. Linking enables multiple applications to share common data. Changes to common data are reflected to all sharing applications whenever the data are changed**.**

Embedding allows container objects to have separate copies of the same data. The next service is drag-and-drop, which permits user to add OLE objects by dragging and dropping them into their containers. Automation service is an OLE services that allow developers to reuse existing components for building a new application by exposing their operations.

2.5 Common Object Request Broker Architecture (CORBA): CORBA is the acronym for Common Object Request Broker Architecture, OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA based -program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. CORBA is

a product of an industry consortium of 500 odd companies called the Object Management Group (OMG). It is a set of specifications for providing interoperability and portability to distributed object oriented applications. CORBA-compliant applications can communicate with each other regardless of location, implementation language, underlying operating system and hardware architecture.

CORBA is composed of five major components:

i. Object Request Broker (ORB): The ORB is the object bus. It provides the middleware that mediates the interactions between client applications needing services and server applications capable of providing them.

ii. Interface Definition Language (IDL): IDL provides architecture and implementation independence to CORBA-compliant applications. It is a declarative language in which object interfaces are defined and advertised. The interface definition is independent of the actual object implementation

iii. Interface Repository: It is an on-line database of object definitions that can be queried at run-time for dynamic method calls, for locating potentially reusable software components and for type checking of method signatures.

iv. Object Adaptor: It provides the run-time environment for the server application and handles incoming client calls.

v. CORBA Services: These augment the functionality of the ORB. CORBA defines services for persistence, transactions, concurrency, database queries, licensing, etc. (Refer figure four)

3. Distributed Artificial Intelligence:

Artificial Intelligence research is fundamentally concerned with the intelligent behavior of machines. In attempting to create machines with some degree of intelligent behavior, AI researchers model, theorize about, predict, and emulate the activities of people. Because people are quite apparently social actors, and also because knowledgeable machines will increasingly embedded in organizations comprising people and other machines, AI research should be concerned with the social dimensions of action and knowledge as a fundamental category of analysis. But current AI research has been inadequate in dealing with much human behavior and many aspects of intelligence.

In most contemporary AI research and practice, the unit of analysis and of development is a computational process with a single locus of control, focus of attention, and base of knowledge-a process organization inherited from von Neumann computer architectures and from psychology. While it is becoming easier to implement such a process as a concurrent system using an underlying distributed processing layer or a parallel language (such as concurrent prolog or lisp) the basic mechanisms of reasoning and problem solving generally remain bound to a single, monolithic conception of knowledge and action. Recently, however, there has been a revival of interest in approaches to analyzing and developing intelligent "communities" which comprise collections of interacting, coordinated knowledge-based processes. The body of research that deals with this problem-level concurrency in AI systems has come to be known as distributed artificial intelligence (DAI). Researchers in DAI are concerned with understanding and modeling action and knowledge in collaborative enterprises. DAI research provides a very rich ground for re-examining some of the premises and formalisms upon which notions such as representation and reasoning, or knowledge and action, are classically located. Contemporary research in DAI inherently deals with social conceptions of knowledge and action (actually, interaction).

Here we briefly examine contemporary research in "classical" DAI. Following that, we introduce some principles which are desiderata for an inherently social conception of knowledge and action, consistent with the premises of open DAI systems.

There are many reasons for wanting to distribute intelligence or cope with multi-agent systems. In some domains, (e.g., distributed sensing, medical diagnosis, air-traffic control), knowledge or activity is inherently spatially distributed. The

distribution can arise. because of geographic distribution coupled with processing or data bandwidth limitations, because of the natural functional distribution in a problem, because of a desire to distribute control (e.g., for fail-soft degradation), or for modular knowledge 9 acquisition. Other reasons for distribution include adaptability, reduced cost, ease of development and management, increased efficiency or speed, history, needs for isolation or autonomy, naturalness, increased reliability, resource limitations, and specialization. Opportunity is a second reason for studying DAI systems. Hardware and software mechanisms for distributing and controlling the interaction of multiple processes have begun to reach maturity, in both shared- memory and distributed-memory multi-computer ensembles. Third, there is interest in integrating existing AI systems to gain power and to leverage capability, which necessarily means coping with problems of discrepancies in representation and design. Fourth, problems are sometimes simply too large or complex to solve by single processes, for reasons of semantic representation as well as computational power; distributed approaches may provide solutions. Finally, it is an empirical observation that most1 human activity involves more than one person. As researchers have tried to understand and model human problem solving and intelligent behavior, they have begun to take this observation more seriously as a foundation for theories.

Research in DAI promises to have wide-ranging impacts in basic AI research (problem representations, epistemology, joint concept formation, collaborative reasoning and problem solving), cognitive science (e.g., mental models, social cognition), distributed systems (reasoning about knowledge and actions in distributed systems, architectural and language support for DAI), the engineering of AI systems: ("cooperating expert systems," distributed sensing and data fusion, cooperating robots, collaborative design problem solving, etc.), and human-computer interaction (task allocation, intelligent interfaces, dialogue coherence, speech acts). As Nilsson has pointed out, DAI research is attractive for fundamental reasons: to coordinate their actions, intelligent agents need to represent and reason about the knowledge, actions, and plans of other agents. DAI research can help to improve techniques for representing and using knowledge about beliefs, action, plans, goals, etc., as well as helping us to discover the extent to which, when analyzed from the outside in-from the social to the individual-these concepts are useful or necessary.

4. Intelligent Agents and Multi Agents:

In this section, we shall focus on on multi-agents and learning, that is, on learning that relies on or even requires the interaction among several intelligent agents. An agent is commonly understood as a computational or natural entity that can be viewed as perceiving in and acting upon its environment, as being autonomous in that its behavior is at least partially determined by its own experience, and as pursuing goals or carrying out tasks (see, e.g., Huhns & Singh, 1998) for a contemporary collection of articles on agents and multiagent systems). Multiagent learning emerged as a topic of active research in the late 1980s and early 1990s, and since then has attracted steadily increasing attention in both the multiagent systems and distributed artificial intelligence community (e.g., Bond & Gasser, 1988; Gasser & Huhns, 1989; Huhns, 1987; O'Hare & Jennings, 1996) and the machine learning community. This attention can be attributed to two primary insights:

i. There is a strong need for learning techniques and methods in the area of multiagent systems. These systems show several characteristics that make it particularly difficult to specify them correctly and completely: for instance, there is no global system control, each agent usually has just incomplete information, the information owned by different agents can be contradictory, and typically the agents are intended to operate in complexopen, large, dynamic, and unpredictable-environments. Because of these characteristics, it is obviously desirable that the agents themselves are capable of improving their own behavior, in addition to the overall system's behavior.

ii. The machine learning area can profit from an extended view capturing both singleagent and multiagent learning. It is one of the primary concerns of this area to understand the principles and mechanisms of learning, whether it occurs in computational or natural systems. Achieving such an understanding requires considering potential learners not just as "stand-alone entities" that act in isolation, but also as "social entities" that interact with one another. This obviously holds for humans and other animals as it lies in their very nature to live and act together, as well as for computing systems as they become more and more connected with each other through long-range and local-area networks. Compared to single-agent learning, multiagent learning raises several qualitatively new 11 issues centered around the relationship between learning and interaction. These issues can be divided into two groups:

i. The role of interaction for learning. Interacting agents, as they exchange information or modify the shared environment in which they are embedded, can significantly influence each other in their individual learning. Possible forms of influence are, for instance, initiation, acceleration, redirection, and prevention of another agent's learning process. Interaction makes it possible that learning by one agent can considerably change the conditions for learning with which other agents have to cope. In particular, interaction is the key to various forms of collective learning in which several agents try to achieve as a group what the individuals cannot, by sharing the load of learning and by pursuing a common learning goal on the basis of their diverse knowledge, capabilities, experience, preferences, and so forth.

ii. The role of learning for interaction. Several dimensions of multiagent interaction can be subject to learning. These include: when to interact, with whom to interact, how to interact, and what exactly the content of interaction should be. An important pattern of multiagent interaction is coordination, among both cooperative and competitive agents. Many learning approaches to coordination are possible. For instance, agents can learn to predict the behavior of others, they can learn to detect and resolve conflicts among their planned activities, they can learn to use a common ontology, they can learn to develop shared viewpoints and assumptions, they can learn to form organizational structures (usually called teams or groups) that enable them to fulfill their design objectives, and they can learn to re configure their styles of coordination to respond best to environmental changes.

It is clear that these issues do not arise in single-agent contexts. There are differences in both the potential paths and the potential goals of learning in single-agent and multiagent settings, and this justifies our contention that multiagent learning is more than a mere magnification of single-agent learning. Researchers in DAI and multiagent systems have found that knowledge representation and reasoning are different for teams of agents and 12 for societies of agents, than they are for individual agents. A group-a team or society might know something that no individual in the group knows. For example, a majority of the group might prefer chocolate ice cream to vanilla ice cream, but the individuals ought to be aware only of their own preferences. Similarly, learning should be different for teams and societies than for individuals. The extent of a society is not fixed and is not necessarily known to any members. Tasks and goals might not be defined or agreed upon, and measures of their success or satisfaction might also not be agreed upon. In such an environment, coordinated behavior is a challenge, but certainly requires learning, both in individual knowledge and in group knowledge. These are appropriate and still open issues for the machine learning research community.

Conclusion

The increasing sophistication of today's information era poses certain challenges to traditional information technology systems. Intelligent Agents and agent based software technology is rapidly evolving to meet demands of this new information era. However, before agent-based solutions can be routinely and successfully exploited in real world problems, first certain fundamental research and software engineering issues have to be addressed. Apart from this, for implementing multi-agents, some of the challenges include understanding the meaning, authentication, secrecy, and security.

Looking at the pace of developments and progress, it is not unlikely that in this century only, we shall see realistic multi-agent deployments. A good start has already been made by Robocup tournaments. The search of web is being improved ever by deploying soft but intelligent agents on the net. A set of complex computational problems are already being solved on the network named as grid computing albeit with less of intelligence. The time is nor far when such attempts would equal the human feet. Let us just not cross our fingers and wait. Let us join this effort of Intelligent agents and swarm intelligence.

REFERENCE

- Bond, A. H., & Gasser, L. (1988) (Eds.). Readings in distributed artificial intelligence. San Mateo, CA: Morgan Kaufmann.
- Gasser, L., & Huhns, M. N. (1989) (Eds.) Distributed artificial intelligence II. London: Pitman.
- Huhns, M. N. (1987) (Ed.). Distributed artificial intelligence. London: Pitman.
- Huhns, M. N., & Singh, M. P. (1998) (Eds.). Readings in agents. San Francisco, CA: Morgan Kaufmann.
- O'Hare, G. M. P., & Jennings, N. R. (1996) (Eds.). Foundations of distributed artificial intelligence. New York: Wiley.
- Sen, S. (1996) (Ed.). Adaptation, coevolution and learning in multi agent systems (Technical Report SS-96-01). Menlo Park, CA: AAAI Press.
- Sen, S. (1997) (Ed.). Multiagent learning (Technical Report WS-97-03). Menlo Park, CA: AAAI Press.
- Sen, S. (1998) (Guest-Ed.). Special issue on Evolution and Learning in Multiagent Systems, International Journal of Human-Computer Studies, 48 (1).
- Weiss, G. (1997) (Ed.). Distributed artificial intelligence meets machine learning. Lecture Notes in Artificial Intelligence, Vol. 1221. Berlin: Springer- Verlag.
- Weiss, G. (1998) (Guest-Ed.). Special issue on Learning in Distributed Artificial Intelligence Systems, Journal of Experimental and Theoretical Artificial Intelligence, 10 (3).
- Weiss, G., & Sen, S. (1996) (Eds.) Adaption and learning in multi-agent systems. Lecture Notes in Artificial Intelligence, Vol. 1042. Berlin: Springer- Verlag.

FIGURE : ONE

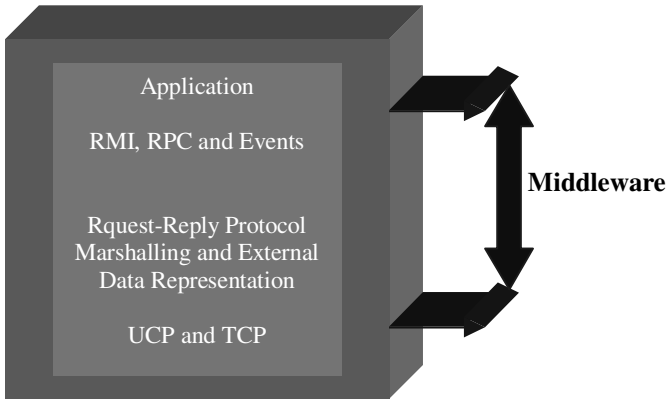

FIGURE TWO : TWO PROCESS COMMUNICATION BY SOCKET

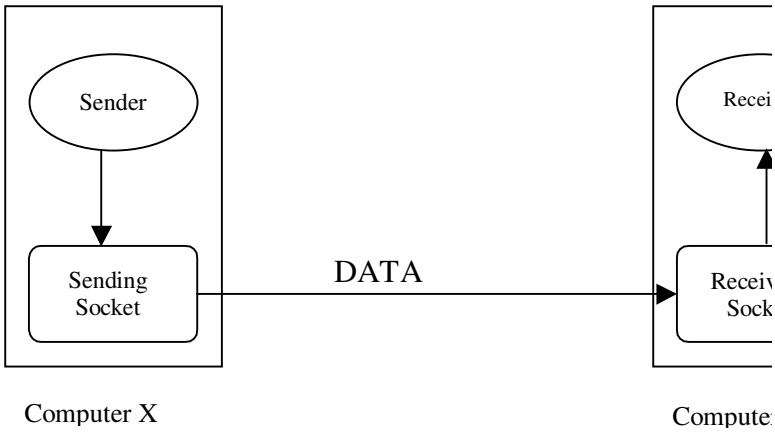

FIGURE : THREE

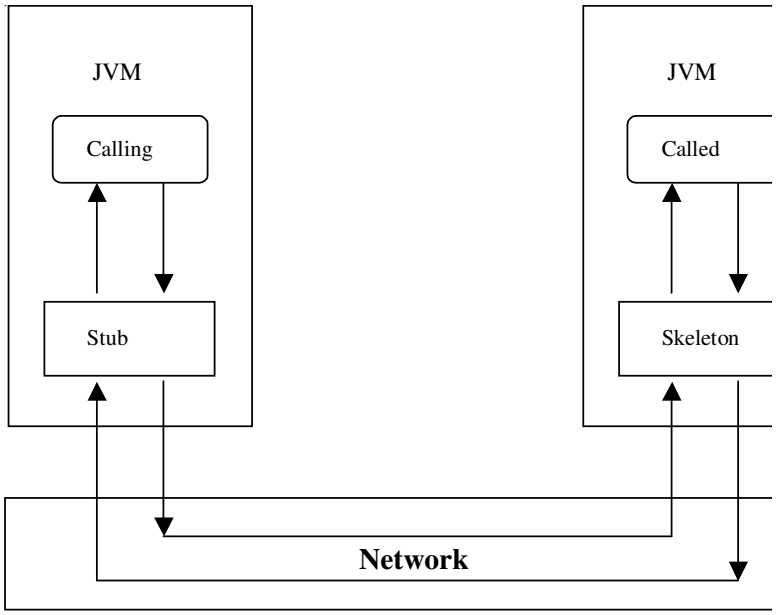

FIGURE : FOUR

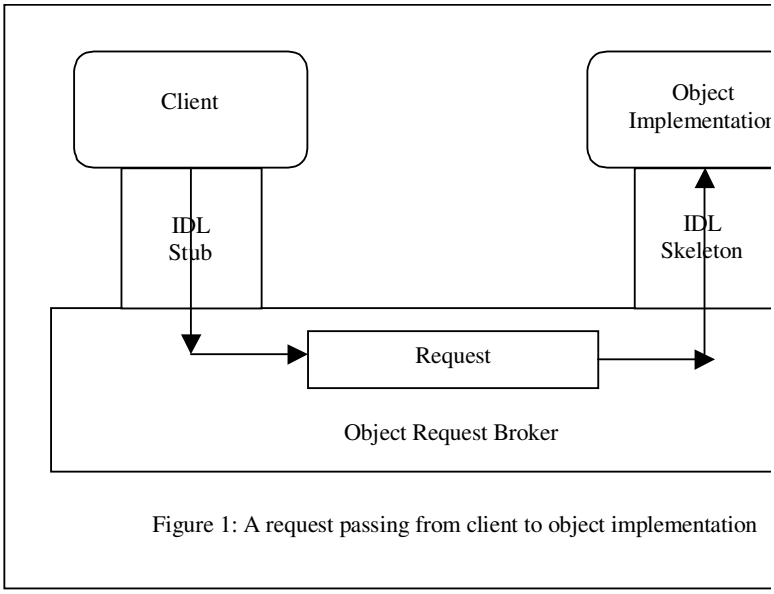

Figure 1: A request passing from client to object implementation