A novel approach of MAS with AI as an AOP in Software Perspective

V. Sree Rekha
Department of Computer Science
Sri DurgaMalleswara Siddhartha MahilaKalasala
Vijayawada, India
rekha.sdmsmk@gmail.com
Dr. R. Padmavathy
Department of Commerce
Montessori MahilaKalasala
Vijayawada, India
padmavathi.raavi@gmail.com

Abstract:Innovative systems are to be designed to support few characteristics such as flexibility and dynamic features due to the increase of number of social computational systems in unpredictable environments. It provides a set of concepts, and a way to think about the world in terms of those concepts. AOP is a more recent development, and still an area of considerable research and standardisation for constructing such systems there is need to seek high-level abstraction to manage the complexity of the systems. Agent-oriented programming (AOP) mainly focus on the construction of multi-agent system (MAS) with the combination of artificial intelligence (AI) and distributed AI. The principles and practices of programming and software engineering are integrated as a common approach to the programming. based on concepts on AI are inadequate for developing dynamic and flexible MAS in open environment. This paper proposes a novel AOP approach, from a software engineering perspective it integrates organizational concepts and mechanism into a language of AOP. The proposed approach consists of a programming model and a corresponding programming language where the syntax and formal operational semantics. Dozens of AOP languages have been proposed in the past two decades. However the acceptance and adoption of AOP in software engineering community remain limited. The paper aims at providing a survey of AOP from software engineering perspectives, including its research history and the state-of-The-Art of researches on agent-oriented programming concepts and models, languages, CASE tools and running manners. It also deals the design principles of programming language such as simplicity, efficiency, regularity, maintainability and reliability. The researchers are implementing the current technology in the software organization's also for the development of certain tools which help the end users to use the various types of features required in the present day world. The artificial intelligence is helpful for development of AOP in the field of computer science which makes the users know about the innovative approach and respond to the real world perspectives in day today life on behalf of software in the organization.

*Keywords:*Flexibility, unpredictable, standardization, AOP, MAS, AI, perspective, maintainability.

I.INTRODUCTION

The main goal of this paper is to provide an overview of the rapidly developing area of software agents serving as a reference point to a large body of literature and to present the key concepts of software agent technology, especially agent languages, tools and platforms. Special attention is paid to significant languages designed and developed in order to support implementation of agent-based systems and their applications in different domains. The metaphor of "intelligent software agents" as basic building blocks for the development of new generation intelligent software systems triggered both theoretical and experimental computer science research aiming to develop new programming languages for agent systems. Few years ago software agent technology has been recognized as a rapidly developing area of research and one of the fastest growing areas of information technology. Agent oriented technologies, engineering of agent systems, agent languages, development tools and methodologies are an active and emergent research area and agent development is getting more and more interesting. There are many approaches, theories, languages, toolkits, and platforms of different quality and maturity which could be applied in different domains. Agents, agent-oriented programming (AOP), and multi-agent systems (MAS) introduce new and unconventional concepts and ideas. Still, there is a number of definitions of the term 'agent' that include a property common to all agents: agent acts on behalf of its user, as well as a lot of additional properties: agent communicates with other agents in a multi-agent system; acts autonomously; is intelligent; learns from experience; acts proactively as well as reactively; is modelled and/or programmed using human-like features (beliefs, intentions, goals, actions, etc.); is mobile, and so on. After more than two decades of scientific work in the field, the challenge is to include agents in real software environments and widely use the agent paradigm in mainstream programming. One way to facilitate this is to provide agent-oriented programming languages, tools and platforms. Our recent overview of the agent programming literature revealed a number of trends in the development of agent programming languages. These trends follow the main achievements of computer science disciplines that are traditionally directly connected to multi-agent systems, i.e. formal methods, object-oriented programming, concurrent programming, distributed systems, discrete simulation, and artificial intelligence.

II.SOFTWARE AGENT CLASSIFICATION AND ARCHITECTURE

Agents are typical inhabitants of open systems like the Internet. An agent is a computer system that, in addition to having the properties identified in the definition of weak agent, is either conceptualized or implemented using concepts that are more usually

applied to humans (knowledge, obligations, beliefs, desires, intentions, emotions, human-like visual representation, etc.)." The "strong notion of agency" corresponds to the usage in the field of artificial intelligence (AI). General classifications in the agent community distinguish between reactive architectures and deliberative architectures. Reactive architectures are considered as simple controls, while deliberative architectures implement complex behaviour including mental attitudes (goals etc.) and planning, based on symbolic representations and models. Hybrid architectures are combinations of both reactive control on the "lower level" for fast responses and deliberative control on the "higher level" for long-term planning. A technologically better classification addresses the possible "states" of an agent. A state is a snapshot of the system (e.g. the content of the memory) at a certain time point on a discrete time scale. Transitions describe the state changes between two time points. For agents, the time scale is usually chosen according to the sense-think-actcycle. This cycle consists of − processing incoming information ("sense", e.g. parsing messages from other agents, analyzing human requests, possibly in natural language etc.) − more or less complex decision procedures ("think", e.g. by simple decision or rules, or by deliberation, planning etc.) − sending outgoing information ("act", sending messages to other agents, preparing answers in a human-like style etc.). An internet agent may in one cycle get a customer request, update its database and send an answer. The state is the content of the database at the end of this cycle. An agent may have different cycles at different time scales: A search engine may answer search requests more frequently than its index machine gets updated. "This creates a need for synchronization efforts which can be facilitated e.g. by different layers." A behaviour (architecture) is called stimulus-response behaviour if there are no different states at all. The response of the agent to an input is always the same (if there is some probabilistic component, then the distribution is the same). It can be produced e.g. by a fixed set of rules, by an input-output table or by a neural network. It doesn't matter if the response routine is a simple or a complex one. A search engine may perform extensive search over large databases and a lot of effort to rank the results. If nothing is stored after answering, then the same calculations yielding an identical answer will be performed every time for the same request. If the responses of an agent to identical requests are different, then they depend on the state of the agent. The search engine may maintain profiles of its users such that the answers depend on the stored profiles. The profile is updated every time the user makes a request, i.e. the state of the agent is changed. It is useful to distinguish between different kinds of states according to their contents: The so-called belief or world model stores internal representations about the situation in the environment for later usage. It is updated according to the sensor inputs. It is called belief because it needs not to be true knowledge (e.g. the profile of a user calculated only by the available user inputs needs not represent her or his true preferences). Future directed states are created by "mental attitudes" related with decision processes. They are named as goals, plans etc., and they guide the future deliberation and actions (towards a formerly chosen goal). A trading agent may have the goal of an optimal transaction. For that, it may develop a plan for searching appropriate offers from databases and for negotiation with other agents. As introduced above, agents are called "stimulus-response" if they do not have states. Nevertheless, their decision procedures might be very complex, e.g. in complex information systems. Agents with states may have a world model (belief) and/or future-directed state components like goals and plans. Their deliberation process considers updates of the world model and commitment procedures for selecting goals and constructing plans. Actually, it is up to the programmer to decide if mental notions are used for data constructs. Sometimes, very simple agents come with mental attitudes. There is nothing wrong with it if it helps for better understanding.

III. ROBOTS AND SOFTWARE AGENTS

The sensors and actuators of the robot provide the input and output for the agent. Robots are often considered as hardware controlled by a software agent acting as the brain. This works well in simple settings, but it poses problems for more complex robots in real environments. Control of such robots is more than information processing: parts of such robots coordinate not only by messages, but by physical interactions too. It is very difficult or even impossible to model the physical dependencies in terms of information processing. However, those relations can be used directly by clever design. Modern robotic approaches are inspired by biology and use local sensor-actor loops, etc. "The key observation is that the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known. The trick is to sense it appropriately and often enough." Related robot controls are able to perform surprisingly complex tasks. Their behaviour emerges from the physical situationless of an embodied entity. The robots have bodies and experience the world directly - their actions are part of a dynamics with the world and have immediate feedback on their own sensations. − Intelligence - They are observed to be intelligent - but the source of intelligence includes: computational engine, situation in the world, the signal transformations within the sensors, and the physical coupling of the robot with the world. − Emergence - The intelligence of the system emerges from the system's interactions with the world and from sometimes indirect interactions between its components. A behaviour is implemented as a simple state machine with few states. Robots built from connected behaviours are capable of performing some complex tasks with relatively simple programming. The behavioural agent architectures are sometimes considered as prototypes of reactive architectures. While single behaviours are simple, their hierarchical combination into a more complex behaviour becomes more and more complicated. Because of that, the hybrid architectures combine low-level approaches with classical reasoning approaches in hierarchical architectures. Such architectures may consist of several levels, where low levels can use behavioural architecture, and higher levels are usually deliberative (symbolic) ones. Symbolic modelling becomes necessary when sensing does not give enough information, or when planning is really needed. Because of the "physics in the loop", the assumptions usually connected with software agents are not fulfilled: Physical components do not behave like objects (or agents). This difference is recently stressed by the investigation of so-called Cyber Physical Systems which are considered as distributed systems where the components perform information processes as well as physical processes. The interaction between the components is of physical and computational nature, as well.

IV. FEATURES OF AGENTS

Depending on different usages of agents, they can have a lot of different features. Such features are often used for classification as well. We have already discussed different basic architectures. Next we describe mobility, size intelligence and ability to adapt and to learn. Relations to other agents are described later in the section on Multi-Agent Systems.

Mobility - Agents can be static or mobile. Static agents are permanently located at one place, while mobile agents can change their location. When a static agent wants some action to be executed at a remote site, it will send a message to an agent at that location with the request for the action. In a similar situation, a mobile agent would transmit itself to the remote site and invoke the action execution. There are a lot of benefits from usage of mobile agents but if we wanted to get all of these benefits without a mobile agent, we would need a large amount of work and it would be practically almost impossible. The advocated utility of mobile agents is to support optimization of sophisticated operations that may require strong interactivity between components or special computing facilities as encountered e.g. in negotiation, network management and monitoring, and load balancing for scientific computing. Mobility of software agents is closely related to the problem of code mobility in distributed programming with applications in operating systems and computer networks. Some problems related with mobile agents concern security and safety. A good overview of code mobility paradigms can be found in the reference paper.

Size and Intelligence - Agents can be of various sizes and can possess various amounts of intelligence. Generally, intelligence of a software agent is proportional to its size, so we can distinguish: big-sized, middle-sized and micro agents. It is difficult to make clear boundaries among these categories.

1. A big-sized agent occupies and controls one or more computers. It possesses enough competence to be useful even if it acts alone, without the other agents in a MAS. A big-sized agent can be as big and as intelligent as an preservation with competences for expert problem solving, e.g. distributed medical care or plane ticket reservation.

2. A middle-sized agent is the one that is not useful without the other agents in a MAS or without additional software. However, it is able to perform some non-trivial task(s). A user-interface agent that acts without other agents and performs some simple actions can also be classified as a middle-sized agent. Mobile agents are usually middle-sized agents.

3. Micro agents (also called the Society of Mind agents) do not possess any intelligence. Minsky followed the idea that the intelligence emerges as a global effect of the overall activity of many simple and unintelligent agents. Adaptation – Adaptive agents can adapt their behavior to different situations and changes in the environment. For example, a navigation system can adapt to changes in traffic (e.g. a traffic jam) and propose alternative routes. This makes adaptive agents more robust to non-predicted changes in a dynamic environment.

Learning - Agents can use learning capabilities for better performance. Learning can be done online, e.g. by data mining from data which are constantly collected through interaction with users (e.g. for profiles). Offline learning refers to training processes (e.g. for pattern recognition) prior to productive agent usage. Agents may possess many features in various combinations.

IV. MULTI AGENT SYSTEMS AND AGENT COMMUNICATION

"Distributed Problem Solving" is performed by agents working together towards a solution of a common problem (e.g. for expert systems). Multi Agent Systems (MAS) take a more general view of agents which have contact with each other in an environment (e.g. the Internet) The rules of the environment as well as the agent controls determine the form of coordination. The agents may be cooperative or competitive. Relations between local and global behaviour in such MAS have been studied using game theory and social theories. Communication via exchange of messages is the usual prerequisite for coordination. Nevertheless, cooperation is possible even without communication, by observing the environment. The two most important approaches to communication are using protocols and using an evolving language.. Both have their advantages and disadvantages. For industrial applications, communication protocols are the best practice, but in systems where homogeneous agents can work together, language evolution is the more acceptable option. Agent Communication Languages (ACLs) provide important features like technical declarations (sender, receiver, broadcasting, peer-to-peer, …), speech act (query, inform, request, acknowledge,…) and content language (e.g. predicate logic). Together with these features, related protocols are defined to determine the expected reactions to messages (e.g. an inform message as an answer to query message). A number of languages for coordination and communication between agents was enumerated. BlackboardBlackboard systems provide a common active database (the blackboard) for information exchange. MAS with many agents are often used for simulations to study Swarm Intelligence and for social simulations in the field of Socionics. Social simulations include simulations of financial markets, traffic scenarios, and social relationships. Swarm intelligence can lead to complex "intelligent" behaviour which emerges from the interaction of very simple agents, e.g. in ant colonies or in trade simulations. Complex problems, e.g. the well-known travelling salesman problem can be solved with swarm techniques. An essential component of agent-based technology and implementation of agent-based systems is a programming language. Such a language, called an agent-oriented programming language, should provide developers with high-level abstractions and constructs that allow direct implementation and usage of agent-related concepts: beliefs, goals, actions, plans, communication etc. Most agent systems are still probably written in Java and C/C++. Although traditional languages are not well-suited for agent systems, it is achievable to implement them in Pascal, C, Lisp, or Prolog languages. Typically, object-oriented languages (Smalltalk, Java, or C++) are easier to use for realization of agent systems as agents share some properties with objects such as encapsulation, inheritance and message passing but also differ definitely from objects vis-à-vis polymorphism. Apart from these standard languages, several prototype languages for implementing agent based systems have been proposed to support better realization of agent specific concepts. Devising a sound classification and analysis methodology for agent programming languages is a very difficult task because of the highly dimensional and sometimes interdependent heterogeneous criteria that can be taken into account, e.g. computational model, programming paradigm, formal semantics, usage of mental attitudes,

architectural design choices, tools availability, platform integration, application areas, etc. Therefore, here we take a more pragmatic approach by firstly proposing a light top-level classification that takes into account those aspects that we consider most relevant for agent systems, i.e. the usage of mental attitudes. According to this classification we find: agent-oriented programming (AOP) languages; belief-desire-intention (BDI) languages, hybrid languages (that combine AOP and BDI within a single model), and other (prevalently declarative) languages. Understanding the current state of affairs is an essential step for future research efforts in the area of developing agent-oriented programming languages.

V. AGENT – ORIENTED PROGRAMMING MODEL

The term Agent-oriented Programming (AOP) was coined in [90] to define a novel programming paradigm. It represents a computational framework whose central compositional notion is an agent, viewed as a software component with mental qualities, communicative skills and a notion of time. AOP is considered to be a specialization of object-oriented programming (OOP), but there are some important differences between these concepts. Objects and agents differ in their degree of autonomy. Unlike objects, which directly invoke actions of other objects, agents express their desire for an action to be executed. In other words, in OOP the decision lies within the requesting entity, while in AOP the receiving entity has the control over its own behaviour, by deciding whether an action is executed or not. Also, agents can often have conflicting interests, so it might be harmful for an agent to execute an action request from another agent. An additional difference is flexibility. Agents often exhibit pro-active and adaptive behaviour and use learning to improve their performance over time. The thread of control is the final major difference. While multi-agent systems are multi-threaded by default, there is usually a single thread of control in OOP. An important part of the AOP framework, as described in a programming language. Agent-orient Programming Language (APL) is a tool that provides a high-level of abstraction directed towards developing agents and incorporates constructs for representing all the features defined by the framework. Most of all, it should allow developers to define agents and bind them to specific behaviours represent an agent's knowledge base, containing its mental state; and allow agents to communicate with each other. The AOP paradigm was very influential for the further development of agent programming languages, resulting in a number of languages. Tools and Platforms Multi-agent systems are deployed and run over specialized software infrastructures that provide the set of functionalities vital for the existence of a realistic multi-agent application. Seen from the perspective of distributed systems technologies, such infrastructures are placed at the middleware level and they include a collection of software functionalities and services that assure: agent lifetime management, agent communication and message transport, agent naming and discovery, mobility, security, etc. An agent framework is a software infrastructure available as a software library, a language environment, or both, which provides the core software artifacts needed for creating the skeleton of a multi-agent system. A software package that provides the core functionalities for deploying and running multi-agent applications is traditionally known as an agent platform. An agent toolkit is a more complex software infrastructure that allows both the development and deployment of a multi-agent system. It is sometimes known as an agent development environment, because of its expected support for all engineering stages of a multi-agent application from requirements to deployment, maintenance and evolution. Most often, a multi-agent system is deployed and runs on top of an agent platform. If an agent platform is not available, at least an agent framework is usually utilized to create the multi-agent system which is then run on a general purpose middleware platform. Agent code can be programmed either using a general-purpose programming language linking with software libraries available in the agent framework via the framework API, or using one of the agent programming languages. Agent forms can be extremely useful because they considerably simplify the development and deployment of a multi-agent system. There is the option to choose between standardized or not-standardized agent platforms. A standard agent platform is compliant with available standards for software agents. Compliance to standards is important for open systems, i.e. systems that might need to interoperate in the future with other systems that are either not available at the moment when the open systems are being developed or that, even if they are available at the moment, still might change in the future.

VI. ADVANTAGES & APPLICATIONS OF MULTI AGENT APPROACH

An MAS has the following advantages over a single agent or centralized approach:

- An MAS distributes computational resources and capabilities across a network of interconnected agents. Whereas a centralized system may be plagued by resource limitations, performance bottlenecks, or critical failures, an MAS is decentralized and thus does not suffer from the "single point of failure" problem associated with centralized systems.
- An MAS allows for the interconnection and interoperation of multiple existing legacy systems. By building an agent wrapper around such systems, they can be inporporated into an agent society.
- An MAS models problems in terms of autonomous interacting component-agents, which is proving to be a more natural way of representing task allocation, team planning, user preferences, open environments, and so on.
- An MAS efficiently retrieves, filters, and globally coordinates information from sources that are spatially distributed.
- An MAS provides solutions in situations where expertise is spatially and temporally distributed.
- An MAS enhances overall system performance, specifically along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse.

Applications of Multi-Agent Research

In artificial intelligence research, agent-based systems technology has been hailed as a new paradigm for conceptualizing, designing, and implementing software systems. Agents are sophisticated computer programs that act autonomously on behalf of their users, across open and distributed environments, to solve a growing number of complex problems. Increasingly, however, applications require multiple agents that can work together. A multi-agent system (MAS) is a loosely coupled network of software agents that interact to solve problems that are beyond the individual capacities or knowledge of each problem solver. MAS applications cover a variety of domains, including

- aircraft maintenance
- electronic book buying coalitions
- military demining
- wireless collaboration and communications
- military logistics planning
- supply-chain management
- joint mission planning

VII. CONCLUSION

Software agents are an emergent and rapidly developing field of research. In the last decade, a number of essential advances have been made in the design and development of software agent languages and the implementation of multi-agent systems. Our intention was to enumerate and present essential features and functionalities of selected languages, tools and platforms, instead of judging them. On one side, we can find agent languages useful for building software agents that can be used as building blocks for the development and deployment of complex distributed applications, usually based on agent or other suitable middleware platforms. On the other hand, we can find agent programming languages used for designing and running complex simulation models that employ the agent metaphor for modelling and simulation of complex systems. However, these languages are not immediately useful for developing real systems, but are rather mostly employed for research in understanding complex systems using agent-based modelling and simulation tools, as agent simulation languages. Most of the recently developed languages find their place in real environments and have been used in developing different kinds of applications. For the development and deployment of a multi-agent system in real environments it is necessary that appropriate software infrastructures exist. Unfortunately, only few of them are still currently available, others either becoming obsolete or not being developed anymore. Furthermore, this prominent technology inspired some authors to go a step further. Finally, it is important to mention that in order to be accepted by the industrial community, MAS applications need to be successfully demonstrated in complex real world pilot systems.

VIII.REFERENCES

1. Agents mailing list, agents@cs.umbc.edu.
2. Ancona, D., Mascardi, V., Hübner, J.F., Bordini, R.H.: Coo-AgentSpeak: Cooperation in AgentSpeak through Plan Exchange, In Third International Joint Conference on Autonomous Agents and Multiagent Systems, Vol. 2, pp. 696 – 705 (2004)
3. Ancona, D., Mascardi, V.: Coo-BDI: Extending the BDI Model with Cooperativity, In Declarative Agent Languages and Technologies, Vol. 2990, pp.109-134 (2004)
4. Azarmi, N., Thompson, S.: "ZEUS: A Toolkit for Building Multi-Agent Systems", Proceedings of fifth annual Embracing Complexity Conference, Paris, (2000)
5. Badano B.M.I., A multi-agent architecture with distributed coordination for an autonomous robot, PhD theses, University of Girona, (October 2008)
6. Badjonski, M., Ivanović, M.: "Multi-agent System for Determination of Optimal Hybrid for Seeding", Proceedings of EFITA '97 - First European Conference for Information Technology in Agriculture, Copenhagen, Denmark, June 15-18, pp. 401-404. (1997)
7. Badjonski, M., Ivanović, M., Budimac, Z.: "Possibility of using Multi-Agent System in Education", Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Orlando, Florida, USA, October 12-15, pp. 588-593. (1997)
8. Badjonski, M., Ivanović, M., Budimac, Z.: "Software Specification Using LASS", Proceedings of Asian'97, Lecture Notes in Computer Science Vol 1345, SpringerVerlag, Kathmandu, Nepal, pp. 375-376. (1997)
9. Badjonski, M.: Adaptable Java Agents – a Tool for Programming of Multi-Agent Systems, PhD thesis, Department of Mathematics and Informatics, Faculty of Natural Science, University of Novi Sad (2003)
10. Barbuceanu, M., Fox, M.S., The architecture of an agent building shell. Intelligent Agents II, LNAI 1037, Spinger-Verlag, pp. 235-250 (1996)

11. Fox M.S., Barbuceanu M., Teigen R.: Agent-Oriented Supply Chain Management, International Journal of Flexible Manufacturing System, vol 12, pp. 165-188. (2000)
12. Bădică, C., Manufacturing and Control: Putting Agents to Work, IEEE Distributed Systems Online, vol. 8, no. 6, pp. 5, (2007)
13. Bădică, C., Ganzha, M., Paprzycki, M.: Developing a Model Agent-based Ecommerce System. In: Jie Lu, Guangquan Zhang, and Da Ruan (eds.): E-service Intelligence, Studies in Computational Intelligence, Volume 37, Springer, 555-578 (2007)
14. Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R.: METATEM: A Framework for Programming in Temporal Logic, In: Proceedings on Stepwise refinement of distributed systems: models, formalisms, correctness, REX workshop, LNCS Volume 430, pp. 94-129 (1990)
15. Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R.: METATEM: An introduction. Formal Aspects of Computing 7(5), pp. 533–549 (1995)
16. Bratman, M.E.: Intention, Plans and Practical Reason. Harvard University Press, 1987.
17. Brazier, F., Cornelissen, F., Gustavsson, R., Jonker, C.M., Lindeberg, O., Polak, B., Treur, J.: A Multi-Agent System Performing One-to-Many Negotiation for Load Balancing of Electricity Use. In: Electronic Commerce Research and Applications Journal, vol.1, no.2, pp. 208-224, Elsevier, (2002)
18. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE, John Wiley & Sons (2007)
19. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE: A software framework for developing multi-agent applications. Lessons learned, Information and Software Technology, Volume 50, Issues 1-2, Elsevier, pp. 10-21. (2008)
20. Bordini, R.H., Dix, J., Dastani, M., Seghrouchni, A.E.F.: Multi-Agent Programming Languages, Platforms and Applications, Springer, (2005)
21. Bordini, R.H., Braubach, L., Dastani, M., Seghrouchni, A.E.F., Gomez-Sanz, J.J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A.: A Survey of Programming Languages and Platforms for Multi-Agent Systems, Informatica, no.30, pp. 33-44 (2006)
22. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason, John Wiley & Sons, (2007)
23. Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (Eds.): Multi-Agent Programming: Languages, Tools and Applications, Springer (2009)
24. Brooks, R.A.: "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, 2(1), pp. 14-23. (1986)
25. Brooks, R.A.: "Intelligence without Reason", Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, pp. 569-595. (1991)
26. Brooks, R.A.: "Intelligence without Representation", Artificial Intelligence, 47, pp 139-159. (1991)
27. Budimac, Z., Ivanović, M., Popović, A.: "Workflow Management System Using Mobile Agents", Proceedings of ADBIS '99, Lecture Notes in Computer Science, Maribor, Slovenia, pp. 169-178. (1999)
28. Bussink, D.: A Comparison of Language Evolution and Communication Protocols in Multi-agent Systems. 1st Twente Student Conference on IT, Track C - Intelligent_Interaction, http://referaat.ewi.utwente.nl/ (2004)
29. Camarinha-Matos, L. M.: Multi-agent systems in virtual enterprises. Proceedings of AIS'2002 – International Conference on AI, Simulation and Planning in High Autonomy Systems, SCS publication, Lisbon, Portugal, pp. 27-36. (2002)
30. Cardelli, L., Gordon, A.D.: Mobile ambients. Foundations of Software Science and Computational Structures, Lecture Notes in Artificial Intelligence 1378, Springer, pp. 140-155. (1998).

Vitae

V. SreeRekha is presently working as Lecturer, Department of Computer Science in Sri DurgaMalleswara Siddhartha MahilaKalasala. She organized 2 Day Workshop on "Internet of Things" conducted by Robokart Innovation Cell, IIT- Bombay. She has participated in 3 International Conferences with online publications and a Research Paper was published in an International Journal with high Impact Factor of 6.577. She participated in 3 National Seminars and 1 International Seminar sponsored by UGC. She also participated in 2 National Workshops sponsored by UGC and participated in 5 Days Faculty Development Programme in Data Science & Big Data Analytics with 97% score in the online examination conducted by ICT – DELL EMC in the Academic Year 2017-2018 respectively. Shehad worked as an Assistant Professor in the Department of Computer Science from June 2008 to May 2017 and had an experience of 9 years in Montessori MahilaKalasala. She participated actively in few National and International Seminars& Workshops sponsored by UGC. Her publications are also published in both National & International Journals.

Dr. R. Padmavathy is presently working as Head, Department of Commerce in Montessori MahilaKalasala Degree College. She Completed 2 Minor Research Projects sponsored by UGC. She participated actively in National (36) and International (10) Seminars. Her publications are also published in both National & International Journals.