ISSN: 2321-2152

IJMECE

International Journal of modern electronics and communication engineering

E-Mail
editor.ijmece@gmail.com
editor@ijmece.com

www.ijmece.com

ISSN2321-2152www.ijmece .com
Vol 6, Issue 3Aug 2018

In-Depth Analysis of Agent-Oriented Software Development

Dr. Prabhu, Imtiyaz Khan,MohdIrshad,

Abstract

One of the most recent additions to Software Engineering is agent-oriented software development. Allowing agents to stand in for high-level abstractions of active things in a software system is one of its many advantages over traditional methods of development. This article provides a review of current literature on industry-strength software engineering, focusing on both generic high-level approaches and on more particular design methodologies.

Keywords: Software Architecture, Components, Design Patterns, UML, and Intelligent Agents

1 Introduction

There is talk of a new paradigm [22] in the study of Software Engineering called Agent-Oriented Software Engineering. However, strong and user-friendly processes and tools must be created before it can become a new paradigm in the software business. First, however, we need define what what an agent is. In computing, an agent (also known as a software agent or intelligent agent) is a piece of autonomous software; the terms "intelligent" and "agent" characterize some of the program's defining characteristics. The word "intelligent" is used because the software is capable of "intelligent behavior," which is "the choosing of actions based on knowledge," and "agent" is used because it explains the program's function. An agent is "one who is allowed to act for or in the place of another," as defined by Merriam-Webster.(1) Virtual characters in video games and simulations (e.g. Quake) Market intermediaries and negotiators (e.g. the auction agent at EBay) 3) Search engine spiders (collecting data to build indexes to used by a search engine, i.e. Google) The weak and strong idea of agency [32] is a typical framework for categorizing agents. Agents under the weak idea of agency have free choice (autonomy), the ability to communicate with one another (social ability), the capacity to react to environmental cues (reactivity), and the power to take the initiative (initiative) (pro-activity). The strong idea of agency has all the features of the weak notion of agency, plus the following: agents are mobile; they tell the truth; they do what they're ordered to do; they're goodhearted; and they act in an optimum way to attain their objectives (rationality).

Department of cse
drprabhu42@gmail.com, imtiyaz.khan.7@gmail.com, mohdirshadislclg@gmail.com,
ISL Engineering College.
International Airport Road, Bandlaguda, Chandrayangutta Hyderabad - 500005 Telangana, India.

Existing agents will be referred to as software agents or agents since they have more characteristics with software than with intelligence.

1.1 Terminology

Since agent-based software engineering is a developing topic of study, I will attempt to define and explain the terminology and connections utilized in academic articles in this area.

When compared to Object-Oriented Programming (OOP), Agent-Oriented Programming (AOP)[29, 30] is generally seen as a step above (OOP). The addition of "Programming" indicates that both ideas are practically at the level of a programming language and its implementation. Shoham first used the phrase "Agent-Oriented Programming" in 1993 [28].

As stated in [8,] Agent-Oriented Development (AOD) is an expansion of Object-Oriented Development (OOD). Even though "Development" might simply mean "Programming," it is more often understood to include the whole development process, from requirements definition and design through the actual coding.

Although all of the following phrases—Software Engineering with Agents [33], Agent-Based Software Engineering [12], Multi-agent Systems Engineering (MaSE) [3, 31], and Agent-Oriented Software Engineering (AOSE) [22, 20, 35, 15]—have the same meaning, the most common use appears to be AOSE. As opposed to AOD, which focuses only on creating an agent-based system, AOSE considers how the system will be used and maintained. However, as was previously said, the term AOD should be avoided for the sake of clarity and to avoid any potential confusion (due to the different interpretations).

All problems with agent-oriented software engineering, as well as those with how and what agents compute, may be summed up under the umbrella term Agent-Based Computing [16].

1.1 Scopeandlimitations

In this article, we will provide a high-level summary of the most up-to-date approaches for creating agent-based systems. An equal amount of attention is paid to broad, overarching approaches and narrower, more focused design processes in the context of software engineering. Specialized agent techniques, such as those developed to enhance agent coordination, cooperation, communication, and artificial intelligence, are thus outside the purview of this work. Jennings et al. [11] and Nwana et al. [27] are cited as helpful background readings that provide broader outlines of the agent research area. Here is how the content of this document is broken down: Aspects of Agent-Oriented Software Engineering are described in Section 2, followed by a description of high-level methodologies in Section 3, followed by a description of design methods influenced by commonly used software engineering methods and standards (such as the Unified Modeling Language, components, and design patterns) in Section 5, which discusses problems, methodologies, and tools for agents in an industrial context.

Agent-OrientedSoftwareEngineering

The primary goals of Agent-Oriented Software Engineering are to develop methods and tools that make it possible to build and maintain agent-based software at a low cost. Moreover, the program must be adaptable, user-friendly, scalable [5] and of good quality. In other words, these problems are quite analogous to those studied in other subfields of software engineering, such as object-oriented program development.

Is there a way to tell living things apart from inanimate ones?

Object-oriented programming (OOP) may be considered as the successor of structured programming [29, 30], while agent-oriented programming (AOP) is an extension of OOP. In object-oriented programming, objects serve as the primary unit of analysis. An object is a collection of related data structures and the procedures that operate on them (functions). In the real world, objects are often used as abstractions for passive things (like a home), whereas agents are often seen as a potential successor to objects since they may enhance abstractions of active entities. Comparable to objects, agents include mental components like beliefs and commitments that may be represented by structures. Also, unlike the ad hoc communications often employed by objects [22], agents enable high-level interaction (using agent-communication languages) between agents based on the "speech act" paradigm, with examples like FIPA ACL and KQML [21].

Another key distinction between agent-oriented programming (AOP) and object-oriented programming (OOP) is that objects are managed from the outside in (whitebox control), but agents exhibit independent behavior that is not immediately manageable by the outside world (blackbox control). That is to say, salespeople may refuse to work with you. [9]

Agents: the answer to all of software's ills?
There is a risk that academics may have unrealistic expectations for the capabilities of agent-oriented software engineering due to the field's youth and fast expansion.

There are risks associated with agent-oriented software engineering, and Wooldridge and Jennings [7, 33] describe some of them. Political, conceptual, analysis and design, agent-level, and societal problems have been identified. Overselling or seeking to apply the notion of agents as the universal answer may lead to political difficulties. Imaginary obstacles might

happen when programmers overlook the reality that agents are software—specifically, multithreaded software. Potential problems with the analysis and design may arise if the developer fails to take into account relevant technologies, such as other software engineering approaches. Too much or too little artificial intelligence in the agent-system might lead to problems at the agent level. Finally, if the developer has a too-idealistic view of agents or uses too few agents in the agent-system, it might have societal consequences.

The issue with all the hoopla
Jennings, a leading researcher in the agent area, notes that the field might easily go the way of the nearly related subject of Artificial Intelligence in the 1980s, which failed to deliver on its promises and became an item of media hype before being "slaughed to death" [16].

2 High-levelMethodologies

Methodologies that use an iterative, top-down approach to designing and building agent-based systems are discussed.

3.1.1 The Gaia Approach
The Gaia technique for agent-oriented analysis and design is presented by Wooldridge, Jennings, and Kinny [10, 8]. Though Gaia is a general methodology that can be applied to both the micro- and macro-levels (agent structure and agent society and organization structure) of agent development, it is not a "silver bullet" solution because it assumes run-time stability in both inter-agent relationships (organization) and agent abilities. Gaia was developed out of a recognition that current approaches fall short of accurately portraying agents' inherent autonomy and problem-solving abilities, as well as adequately modeling agents' idiosyncratic methods of carrying out interactions and constructing hierarchies. With Gaia, developers may methodically create a design that is ready for deployment in light of system requirements.

In Gaia analysis, identifying roles is the first stage, followed by modeling the relationships between those roles. The four components of a role are responsibilities, permissions, tasks, and procedures. There are two sorts of responsibilities: those that contribute positively to the system (liveness qualities) and those that safeguard it (safety properties). The role's permissions define its capabilities and the data it has access to.

One can get into it without any problems. An activity is any job carried out by a role independently of any other roles. Distinct roles, such as a seller, may allow for different auction procedures, such as the "English auction," and they are referred to as "protocols." When it comes to defining roles and the characteristics that go along with them, Gaia provides formal operators and templates, while interaction representations may be modeled using the platform's schemas.

Gaia's design process begins with a step to translate roles into agent categories, followed by steps to generate enough instances of those sorts of agents. The next phase is to figure out what kind of services model is required to perform a certain function in one or more agents, and the last step is to build the acquaintance model to represent the agents' interactions with one another.

Gaia's limitations mean it has limited use for Internet-based applications, but it has shown to be an effective method for building closed-domain agent-systems. Zambonelli, Jennings, et al. [35] offer various adaptations and enhancements to the Gaia technique in order to facilitate the creation of Internet-based applications, which are beyond the scope of the original method's domain limits.

The works of Chaib-draa and others also explore both the micro and macro levels of agent modeling. [2]

Methodology for Multiagent Systems Engineering 3.2
Multiagent System Engineering Approach is recommended by Wood and DeLoach [3, 31].

(MaSE). When compared to Gaia, MaSE is broad and can run on a wide variety of platforms, but MaSE's automated code development features take it a step further. Since there aren't any tried-and-true methodologies or robust toolkits available for developing agent-based systems, MaSE was developed to fill this gap. MaSE's purpose is to guide the designer through the process of creating an agent system, beginning with a system definition. MaSE shares some of Gaia's domain constraints, but it also insists on one-to-one rather than multicast agent interactions.

The steps of the MaSE technique may be thought of as a logical pipeline with seven distinct stages. First, the objectives of the system are captured, which involves translating the original system design into a goal hierarchy. Goals are determined by analyzing the needs stated in the original system specification, and then ranked by importance.

importance in a well-organized, hierarchically-ordered hierarchy based on subject. The second step, called "Applying Use Cases," involves developing the system's use cases and sequence diagrams from the initial specification. As such, use cases depict the logical flow of information between the different roles in a system and the system itself. It is possible to determine the minimal amount of messages required to be exchanged across roles in a system by drawing a sequence diagram. Refining such positions to make them more specific to the first-phase aims is the focus of the third stage. Typically, one role is used to represent a single objective, however sometimes many objectives might be collapsed into a single role. To go along with each position, a list of tasks is formulated, which details the steps to take in order to achieve the role's objectives. State diagrams serve as the primary means of defining tasks. In the last stage, "developing agent classes," a graphic is made to show how various responsibilities are assigned to various agent classes. This diagram is similar to object class diagrams, but instead of focusing on the inheritance of structure, it emphasizes the semantics of dialogue at a higher level. Phase five, "constructing dialogues," involves defining a coordination protocol for interacting agents via the use of state diagrams. Internal agent class functionality is developed in step six, assembly. Belief-Desire-Intention (BDI), reactive, planning, knowledge based, and user-defined agent architectures provide the basis for the selected functionality. When the system design step is complete, real agent instances are created using the agent classes, and the whole thing is laid out in a deployment diagram.

There is hope that in the future MaSE will be able to provide fully automated code creation according to the deployment design.

3.1 Modeling database information sys-tems

When planning IT infrastructure, Wagner [29, 30] recommends using an AOR modeling method. Specifically, AOR is influenced by the Entity-Relationship (ER) meta-model and the Relational Database (RDB) model, two of the most popular database modeling approaches.

The ER meta-model is meant to facilitate the mapping of data-entity relationships to a database-ready information system architecture. This transformation is well-supported for inert entities like objects but falls short when trying to model dynamic actors like agents within an information system, which is why the AOR-model was developed: to supplement the ER-model and allow for the modeling of relations between agents as well as static entities.

In AOR, there are six distinct kinds of entities: agents

,

everything that happens, everything that people do, everything that people claim, and everything that people own. Each party's promises are interpreted as a counterclaim by the opposing party. Groups of people working together are represented as "sub-agents" in the model. Sub-agents may independently take some acts, but they must also undertake obligations for the agent-organization, such as keeping an eye on claims and other relevant events. The services and permissions outlined in the Gaia methodology [10] seem to align with the understanding of responsibilities and rights. Magnanelli et al. [23] provide an example of a DBIS built on an agent-based model.

3 DesignMethods

4.2 Methodologies described here draw heavily on the practices and guidelines established in the area of object-oriented software development.

4.3 To ensure uniformity in the creation of object classes, the Universal Modeling Language (UML) was created as a graphical representation language. Support for constructing sequences, components, etc., in fact all sections of an object-oriented information system architecture, was added subsequently and has substantially expanded its

functionality.

It has been proposed by Yim et al. [34] that multi-agent systems may benefit from an approach to design that centers on the systems' architecture. The technique, which is grounded on common UML extensions based on the Object Constraints Language (OCL), allows for the conversion of agent-oriented modeling issues into object-oriented modeling problems. Instead of the more often used connection types between object classes, such inheritance, the converted relations between agents are employed as relations between object classes in the design process. This approach allows designers and developers to use current UML-based tools and expertise in designing object-oriented systems.

For Agent-Interaction Protocols, Odell, Parunak, and Bauer [14] proposed a three-layer structure (AIP). To begin, AIP are defined as patterns that both describe the message exchange between agents and the related limitations on the content of these messages. Unlike the UML-based design [34] proposed by Yim et al., Odell et almethod .'s requires modifications to both the UML visual language and the articulated semantics. Modifications to the following UML representations are needed for the representation to work properly:

packages, templates, sequence diagrams, collaboration diagrams, activity diagrams, and statecharts are all types of tations. UML packages and tem- plates are used to provide a reusable representation of the communication protocol (i.e., the kind of interaction) at the first layer. The second layer makes use of sequence, collaboration, activity, and statechart diagrams to depict the interactions between agents (i.e., which types of agents may communicate with one another). Activity diagrams and statecharts are used to depict the third layer of agent processing, which explains the agent's motivations and decision-making for each action it does.

To be able to express all characteristics of agents, Odell et al. [13] propose an extension to UML they term Agent UML (AUML). There is a proposal to include AUML into UML 2.0 [17] that has been presented to the UML standards group. Adding greater role definition to UML, as proposed, would need revising the UML sequence diagram structure. The description of the UML package must be altered so that agents, rather than operations, may be represented as interface points. Agents are mobile in the sense that they may freely travel between various agent systems. This requires a modification to the specification of the deployment diagram in UML.

At the agent level, the highest abstraction level in Agent-Oriented Software Engineering, Bergenti and Poggi [15] propose using four agent-oriented UML diagrams. Since no modifications to the UML standard are needed, it is analogous to Yim's method. The first is the UML static class diagram-based ontology diagram, which models the universe in terms of interactions between things. The second is the UML deployment architecture diagram, which is used to depict the setup of a multi-agent system. The third diagram is a protocol diagram, which follows the same conventions as the UML collaboration diagram to depict the interaction language. For reference, below is the first layer of the communication protocol as shown by Odell et al. [14]. Fourth, each agent's capabilities may be represented by a role diagram, which is derived from the UML class diagram.

In order to define and depict social systems in UML, Parunak and Odell [9] integrate exisiting organizational models for agents in a UML-based framework. This work enhances the UML add-ons known as Agent UML.

4.4 DesignPatterns

Design patterns are recurring structures or idioms in software or computer code.

When it comes to design patterns for mobile agents, Aridor and Lange [1] propose a taxonomy. They also provide examples of patterns that may be considered members of each category. The goal is to lessen the time and money needed to create mobile agent systems while improving their reusability and code quality. The three groups in this system are: location, activity, and communication. The forwarding pattern, for example, explains how newly arriving agents might be passed to another host and belongs to the traveling class of patterns since it applies to agents that move between different settings. The patterns in the task class outline the many ways in which agents might carry out their work; for instance, the plan pattern outlines the steps necessary to complete numerous tasks simultaneously on different hosts. It is the job of interaction class patterns to detail

the means through which autonomous entities might coordinate their efforts. The facilitator is an interaction class pattern that specifies a kind of agent that can help other agents discover and be found according to their skillsets.
Rana and Biancheri [26] use Petri Nets to simulate the mobile agent meeting pattern, another approach to mobile agent design patterns.
A seven-layer architectural pattern for agents is proposed by Kendall et al. [6] ([19, 18]), along with sets of patterns that fall under each tier. Mobility, translation, cooperation, action, reasoning, belief, and sensation are the seven tiers. The agent's mental model is picked using patterns in the lowest three layers; for example, if the agent's job is to respond to stimuli, the reactive agent pattern should be chosen, while if the agent's job is to interact with humans, the interface agent pattern should be chosen. The benefits of using patterns in conventional software development are cited to support the decision to use this approach to agent creation.

The layered architecture has a comparable rational categorization of patterns to the one described in the work of Aridor and Lange. The classes of travel (represented by the Mobility and Translation layers), cooperation (represented by the Interaction layer), and activities (represented by the Actions layer) are all represented by the respective layers of this diagram. This method for mobile agents differs from others in that it attempts to include all of the most common varieties of agent design patterns.

4.18 Components
Components are conceptual clusters of linked things that work together to provide a certain set of features. This may seem very similar to agents, however unlike agents, components do not make decisions on their own. Components have proven to be a popular and successful software development strategy because they provide a higher level of re-use than does the construction of single classes from scratch.

A three-tier architecture is proposed by Erol, Lang, and Levy [5] to facilitate the construction of agents by the use of reusable components. The foundation of interactions is laid by the roles and words of the agents involved. The second level consists of the agent's local knowledge and expertise, which is used to save the agent's execution state, plan, and restrictions. The third layer, information content, is passive and often domain-specific due to its frequent usage in encasing outdated systems such as mainframe database applications.

.Agentsinthereal-world
Agent-oriented programming is gaining traction in the business world, but it has yet to catch on to the same extent as object-oriented programming. This section discusses the successful applications of agents in the manufacturing sector, including where and how they have been used.

In a business setting, Parunak [25] provides a definition of agenthood along with a taxonomy and maturity assessment. His goal is to spread knowledge about agent-oriented software engineering and increase its practical use in business.

It is argued that agent-oriented programming, or "agenthood," is nothing more than an incremental enhancement of the tried-and-true approach of object-oriented programming.

Agent systems are placed into one of three categories based on their environment, with digital (including software and digital hardware), social (including human users), and electromechanical settings all being represented (non-digital hardware, e.g. a motor). The agents are then categorized in the taxonomy based on the kind of interface they provide. There is a correlation between interface varieties and ecosystems:

There include digital (such as communication protocols), social (such as user interfaces), and electromechanical (e.g. motor control interfaces).

A maturity meter of agent-based systems is established to be able to quantify the degree of agent technology and systems since few business users, in contrast to academics, are early-adapters of new and immature technology. There are six stages of maturation in the met- ric, from prototypes to finalized goods. The least developed category is modeled applications, which are essentially architectural descriptions or assessments based on theory. Because they are lab simulations, mimicked apps continue to be a rather immature part of the metric. When it comes to software development, prototypes are the next maturity level up; they function in a non-commercial setting yet use actual hardware. While it is reasonable to assume some level of stability from pi-lot applications, they are not considered production-ready until they have been in use for a given amount of time. Many companies are using the program in production, but they need help with setup and upkeep. The most developed applications have reached the point where they can be marketed as goods, packaged in a box, and delivered to a customer's desk; in most cases, a layperson can set them up and keep them running without any special

knowledge or training.

What are the where and how of agents in the business world?
The uses of agents in industry are discussed by Parunak [24]. We take into account the industrial application domains of scheduling, control, cooperation, and agent simulation. Followed by this is a presentation and discussion of various tools, methodologies, insights, and difficulties related to the creation of agent systems.

Scheduling in manufacturing is establishing a sequence for and setting times for various steps in the manufacturing process. The goal is to decrease resource needs per unit and the risk of failures while increasing the number of units produced each time slot without sacrificing product quality. Controlling processes and equipment is essential for ensuring they run on time. Machine power regulation is one kind of control, but more complex cybernetic control of processes in real time is also possible. As an example, throughout the design process, engineers and designers must work together to ensure that goods are both aesthetically pleasing and safe for consumers to use. Since there is a high initial investment for a factory to begin producing electronics, for example, this sector of industry is not attractive to many companies.

The production process must be simulated in a cost-effective manner.

Techniques used by agents in the marketplace
Rockwell's Foundation Technology and DaimlerChrysler's Agent Design for agent- based control [24] are two of the offered methodologies for developing industrial agent systems.

When designing agent-based control architectures, Rockwell's Foundation Technology takes into account four factors: fault-tolerance in a multi-objective setting; self-configuration to support new products and rapidly changing old ones; productivity; how to at least maintain and hopefully improve productivity by applying agents; and equating.

Daimler-Agent Chrysler's Design method is likewise structured in four stages, like Rockwell's. It begins with the analysis and creation of a model of the manufacturing job, continues with the identification and classification of the roles required, moves on to the specification of interactions between roles, and concludes with the specification of agents that will carry out these roles. Regarding role identification and interplay between roles, this technique is quite similar to the Gaia [10] and MaSE [31] approaches.

4 Conclusion

To that end, this work has aimed to provide a synopsis of the state of the art in agent-oriented software engineering in the last several years. Additional research needs to focus on both a more in-depth examination of the field and more rigorous testing and experimentation with the methods.

References

[1] [1] Agent Design Patterns: Elements of Agent Application Design, by Y. Aridor and D. B. Lange. 1998 saw the publication of the proceedings for the second international conference on autonomous agents.

[2] Micro and macro scale relationships in agent modeling [2.] Chaib-draa B. Pages 262-267, 1997, in Proceedings of the First International Conference on Autonomous Agents.

[3] Multiagent Systems Engineering. [3] DeLoach, S. A. Methodology and language for developing agent-based software. Agent-Oriented Information Systems Proceedings, 1999, pages 45–57.

[4] Formalizing Agent-Based System Development with Graph Processes. [4] Depke, R., and Heckel, R. Graph Transformation and Visual Modelling Techniques Workshop (GTVMT'00), in Proceedings of the ICALP'2000 Satellite Workshops, 2000, pp 419-426.

[5] Based on the work of Erol et al. Agent Construction Using Shared Parts. 2000: Pages 76–77 in Proceedings of the Fourth International Conference on Autonomous Agents.

[6] According to [6]E. A. Kendall, P. V. M. Krishna, C. V. Pathak, and C. B. Suresh. Agencies that are smart and mobile and follow patterns. 1998 saw the publication of "Proc. of the Second International Conference on Autonomous Agents," which ran on pages 92–99.

[7] Challenges in agent-oriented software development [7], by M. J. Wooldridge and N. R. Jennings. As published in the proceedings of the 1998 international conference on autonomous agents, pp 385–391.

[8] In [8] Wooldridge M. J., Jenning N. R., and Kinny D. An approach to agent-based research and development. Published in 1999 as pages 69–76 in Proceedings of the Third International Conference on Autonomous Agents.

[9] According to [9] H. V. D. Parunak and J. A UML Model of Social Structures. Published in Proceedings of the 2001 International Conference on Autonomous Agents.

[10] According to [10]M. J. Wooldridge, N. R. Jennings, and D. Agent-oriented analysis and design using the Gaia technique. September 2000 issue of Autonomous Agents and Multi-Agent Systems, volume 3, issue 3, pages 285–

312.
[11] According to [11] N. R. Jennings, K. Sycara, and M. J. Wooldridge. The Future of Agent Research and Development: A Roadmap. First published in 1998, Autonomous Agents and Multiagent Systems, 1(1), pp. 7-38.
[12] When it comes to creating software, [12] Jennings, N. R., "On agent-based software engineer- ing," is where you should look. Information Technology with Artificial Intelligence, Year 2000.
[13] According to [13] J. Odell, H. V. D. Parunak, and B. Bauer. The Expansion of UML for Agents. Proceedings of the Agent-Oriented Information Systems Workshop (AOIS) at the 17th AAAI conference on artificial intelligence (AAAI), 2000.
[14] Represent- ing Agent Interaction Protocols in UML. By Jonathan Odell, Hamid V. D. Parunak, and Brian Bauer. It was held in 2000 during the First International Workshop on Agent-Oriented Software Engineering (AOSE-2000).
[15] Exploiting UML in Multi-Agent System Design. By F. Bergenti and A. Poggi. Workshop on Engineering Societies in the World of Agents (ESAW'00), Proceedings of the ECOOP, 2000, pp. 96–103.
[16] The potential and dangers of agent-based computing [16]. Jennings, N. R. Published in Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), pp 1429-1436.
[17] Suggested UML Extensions for Agents, by John Odell and Christopher Bock, December 1999 [17].
[18] To cite: [18] E. A. Kendall, M. Malkoun, and C. Jiang. Using OOP to create multi-agent systems. In the June 1997 issue of the Journal of Object-Oriented Programming.
[19] With reference to [19] Kendall, E. A., M. Malkoun, and C. Jiang. Agent-based systems as a domain for the application of object-oriented analysis. Article published in the February 1997 issue of the Journal of Object-Oriented Programming.
[20] For more on why an agent-based approach is necessary for developing complex software systems, see [20] Jennings, N. R. forthcoming in Communications of the ACM, 2001.
[21] This is according to [21] Y. Labrou, T. Finin, and Y. Peng, Agent Communication Languages: The Current Landscape. March/April 1999 issue of IEEE Intelligent Systems, volume 14 number 2.
[22] Problems with Agent-Oriented Software Engineering. [22] Lind, J. An Agent-Oriented Approach to Software Engineering: Proceedings of the First International Workshop, AOSE-2000.
[23] Databases for Agents and Agents for Databases. [23]. Magnanelli, M., and Norrie, M. C. June 2000 saw the publication of the proceedings from the second annual international bi-conference workshop on agent-oriented information systems.
[24] A Review of Industrial Agent Applications from the Perspective of a Practitioner. [24] H. V. D. Parunak. In December of 2000, the journal Autonomous Agents and Multi-Agent Systems published issue 3(4), pages 389–407.
[25] Experiences and Issues in the Design and Implementation of Industrial Agent-Based Systems [25]. Parunak, H. V. D. Agents in Overalls. As published in the International Journal of Cooperative Information Systems, Volume 9, Issue 3, pages 209–227, 2000.
[26] Meeting Design Pattern for Mobile-Stationary Agent Interaction: A Petri Net Model, by O. F. Rana and C. Biancheri. This work was published in the proceedings of the 1999 Hawaii International Conference on System Sciences.
[27] Those two authors are [27] Nwana H. S. and D. Ndumu. A new angle on the study of software agents. Reference: 14(2), 1999, The Knowledge Engineering Review, pp. 1-18.
[28] Agent-oriented programming. [28] Shoham Y. The Journal of Artificial Intelligence, Volume 60, Issue 1, Pages 51–92, 1993.
[29] The following is a snippet from: [29] Wagner, G. Agent-Object-Relation Modeling. As presented at the Second International Symposium: "From Agent Theory to Agent Implementation," held in conjunction with EMCRS 2000 in April 2000.
[30] Organizational Information Systems: An Agent-Oriented Approach to Analysis and Design. [30] Wagner, G. May 2000 in Vilnius, Lithuania, during the Fourth IEEE International Baltic Workshop on Databases and Information Systems.
[31] An Overview of the Multiagent Systems Engineering Methodology. [31] Wood, M. F., and S. A. DeLoach. It was held in 2000 during the First International Workshop on Agent-Oriented Software Engineering (AOSE-2000).
[32] To learn more about intelligent agents and how they work, check out [32]. It was published in 1995 in the journal Knowledge Engineering Research and Development, volume 2 issue 10 page numbers 115-152.
[33] The Perils of Agent-Based Software Development, by Michael J. Wooldridge and Neil R. Jennings [33]. May/June 1999 issue of IEEE Internet Computing, pp.20-27.
[34] Source: Architecture-Centric Object-Oriented Design Method for Multi-Agent Systems, by H. Yim, K. Cho, K. Jongwoo, and S. Park. Multi-Agent Systems: Fourth International Conference, Proceedings, ICMAS-2000, 2000.
[35] By F. Zambonelli, N. R. Jennings, A. Omicini, and M. J. Wooldridge. Chapter 13: Coordination of Internet Agents, Models, Technologies, and Applications. It was published by Springer in the year 2000. Software engineering with an emphasis on agents for use in web-based programs
[36].