TOWARD PROGRAMMING PARADIGMS FOR AGENT ORIENTED SOFTWARE ENGINEERING
Salaheddin J. Juneidi
Department of Information and Communication Systems Engineering
School of Sciences
University of the Aegean
Samos, Greece
jneidi@aegean.gr

ABSTRACT
Agent Oriented Software Engineering (AOSE) is defined as a new trend in computing with methodologies and modeling techniques for integrating agent entities in the software system, proposed to answer the increasingly complex distributed, large, open, dynamic, unpredictable, heterogeneous, and highly interactive application environments. Motivated by at least two main fields in computer science, Software Engineering (SE), and Agent Base Computing, with great amount of overlapping and related fields which create confusion and unclear view of a formal description of Agent Oriented System in terms of analysis, design and implementation. This paper helps the readers to clear the confusion toward the future of AOSE through returning to the basic definitions, based on the reflective relation between programming paradigms as the tool of software application and SE approach’s evolutions.

KEY WORDS
MAS/ABS modeling, OOSE, AOSE, programming paradigms.

1. Introduction
Software engineering is a science under continuous development to keep track with the evolutions in programming paradigms, computer resources and needs, for a more “clear” and adequate representation for software systems developing process.

Software engineering is about to design and develop high-quality software. The generic software engineering life cycle begins from requirements and specification analysis, system design and program coding, until it reaches system testing and delivery. The most frequently changing phases of this life cycle are analysis and design phases. These two phases comprise the “core” of the system development process, but still they cannot be separated from the other phases. For instance, “analysis” comes when the requirements have been set, “coding,” reflects the system design and system “testing” is being done according to the requirements gathered in the requirements specification phase.

Agent-Oriented Software Engineering (AOSE) is being described as a new paradigm [1] for Software Engineering. Consequently methodologies with modelling techniques and implementation tools have to be developed to support the integration of software agents into a software system.

A software agent, is a piece of autonomous software [2] or an autonomous entity that can interact with its environment [3]. Agent characteristics that have been used for the classification of agents (e.g. in [4] and [5]) include: Autonomy (agents have their own will), social ability (ability to interact with the environment and other agents), pro-activeness (agents take the initiative to act), adaptiveness/reactiveness (they respond to stimuli from their environment) and mobility (they move around between platforms). Stronger notions of agents attribute them with human-like properties such as character (personality, emotional states and beliefs), abilities to learn and rationality (poses goals, forms assumptions and plans towards achieving goals).

Embedding agents with these characteristics into software may calls for a new programming paradigm. Although the expression “Agent Oriented Programming Language” has already been used [6], but not been clarified or applied, this paper examines the possibility of implementing agents in the most recent programming paradigm defined by the Object Oriented approaches. The paper investigates the aspects that affect programming paradigms at different levels of abstracting agents and objects. Then, identifies the reflective relation between software engineering evolution and programming paradigms development to have a confusion free view toward Agent Oriented Software Engineering (AOSE) from Multi-Agent System (MAS) and Agent-Based System (ABS) modelling. Previous research works implicitly and explicitly referred to all of these titles as to be holding the same meaning [7,8,9,10,11], this paper will differentiate between them and will show that AOSE has different view and applications from MAS and ABS modelling and expresses AOSE is the natural evolution of Object-Oriented Software Engineering (OOSE).

Figure 1: Levels of abstraction that affect programming paradigm
The paper is structured as follows: Section 2 illustrates the levels of abstraction of different programming paradigms, and distinguishes agents from objects at each such level. Section 3 identifies the reflective relation between programming paradigms and software
engineering methodologies and techniques. Based on the outcomes of sections 2 and 3, section 4 discusses the differentiation between AOESE, MAS & ABS modelling. Finally, the paper concludes with a prospective view of agents and objects co-existence.

2.0 Aspects affecting programming paradigms

Programming paradigms are affected by different aspects at different levels of abstraction. Figure 1 shows these levels of computing abstraction. The triangle shape shows which abstraction level dominates the others. For instance, hardware plays a dominant factor on operating systems, while operating systems abstract factors related to hardware, and so on.

2.1 Hardware

Hardware is the most dominant layer. Any evolution at this level of computing naturally affects other layers. Hardware refers to the computer architecture, which didn’t attested many changes since the von Neumann architecture proposed in the late 1940s. Major achievements at this layer include exponential increase of CPU and memory speed and capacity.

2.2 Operating Systems

Advances in hardware resulted in developing advanced operating systems with more effective CPU and memory management. Parallelism, multithreading, sharing and more, increased the efficiency and the utilization of computer resources.

<table>
<thead>
<tr>
<th>Structural Elements</th>
<th>OOP</th>
<th>AOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract class</td>
<td>Generic role</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>Domain specific role</td>
<td></td>
</tr>
<tr>
<td>Member variable</td>
<td>Knowledge, belief</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Capabilities, (complex and primitive) actions</td>
<td></td>
</tr>
<tr>
<td>Relation</td>
<td>Collaboration (uses)</td>
<td>Negotiation</td>
</tr>
<tr>
<td>Composition (has)</td>
<td>Institutionalised agents, groups of agents</td>
<td></td>
</tr>
<tr>
<td>Inheritance (is)</td>
<td>Role multiplicity</td>
<td></td>
</tr>
<tr>
<td>Instantiation</td>
<td>Domain specific role and individual knowledge</td>
<td></td>
</tr>
<tr>
<td>Polymorphism</td>
<td>Service matching</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Differentiation between agents and objects at the programming level

2.3 System Runtime

The runtime system of a particular programming paradigm provides the environment for program interpretation. These environments can be radically different between different paradigms. They may be restricted to administrative tasks such as managing the heap or they may also provide slightly more elaborate services such as garbage collection.

At this level of abstraction, “agents” have distinct behavior from “objects”. In an object-oriented runtime system, the objects are statically represented by the objects’ architecture. This architecture contains the current state of any object and objects’ relations to the object classes, which subsequently determine the operations that can be performed by this object. An object is usually represented as a collection of data elements with associated functions and the granularity of objects is potentially not limited. The object management system is responsible for managing the relations between objects and classes (e.g., the inheritance relation) and for the manipulation of objects (e.g., object creation or destruction). Furthermore, the object management system is also responsible for dynamic aspects, such as method selection of polymorphous objects, exception handling and garbage collection.

In an agent-oriented runtime system, things are distinctly more complicated. Agent architectures are far more complex than the object architecture, especially because of the dynamic aspects that agents deal with. Each agent perceives the state of its environment, integrates the perceived facts in its knowledge base, forms beliefs, desires, goals and intentions to act and finally executes the planned activities (possibly in coordination with other agents).

2.4 Programming Languages

At this level of abstraction, the syntax and semantics of a programming language for the manipulation of entities at the runtime level is defined. The programs that are written in a particular programming language are interpreted at the system run-time level. In the programming language level, as well as it is at the runtime level, there is a differentiation between objects and agents, as it is shown in the following table (Table:1)

2.5 Design Language

Design languages are further abstractions from a particular programming language that aim at the conceptual modeling of a system at a more coarse grained level. Design languages often use graphical notations that make it easier for the designer to use and manipulate the overall system structure.

In the object oriented community UML is a well established design language being supported by case tools such as the Rational Rose Software®. In the agent based world there is not any uniform design language mainly due to the ellipsis of an agent oriented programming paradigm. However, there is a large number of design toolkits for special kind of agent architectures and platforms. Examples for existing agent-based design languages range from source-level frameworks such as SIF and Swarm up to complex and powerful tools such as the ZEUS toolkit from British Telecom that provides drag-and-drop mechanisms for putting together multi-agent applications.

Concluding the above, there is a clear distinction between agents and objects at the system run-time, programming and design language levels. The above made analysis shows the importance of establishing a paradigm for agent programming and agent oriented system design that shall take into account the distinct characteristics of agents and objects at the run-time, programming and design level.
### 3.0 Software Engineering and Programming Paradigms

A programming language is the tool for software building. Historically, as these languages get more advanced, new programming approaches come up. Consequently, new development methods and techniques burst. As an outcome of this process, new software engineering approaches are finally established.

Table 2 defines the reflective relation among software engineering evolutions and the developments in implementation tools (programming languages). For memory refreshment, back since top-down programming, to modular (or structural) programming and lately to object-oriented programming, software engineering has been developed from one part or another to encounter each programming approach in a way that they can be understood and applied by programmers. In other words, software engineering reflects the different programming paradigms, basically because it provides models and techniques for blueprinting software applications developed by computer languages.

Yet a new comer arises in software development with all new characteristics of agency. As pointed above, agents must be recognized in a different way from objects. Considering the previously mentioned characteristics of agents, such as they have their own thread of control; individual rules and goals, making them appear like “active objects with initiative”[12]. However, objects are considered relatively passive compared with agents because their methods are being invoked only when some external entity sends a message to act. Due to all the differences between agents and objects, further development is needed in software engineering and programming languages to adopt software agents, to be applied and represented into software systems.

As discussed in the previous section there are quite differences between objects and agents in terms of their abstractions in run time and all the way down levels that affect programming paradigm. Proposed solutions for integrating agent components within software application that contains agent and non-agent entities can be managed through two methods. The first method is to use a dedicated open source of commercial agent platform with stranded object oriented language. The generic cycle of an agent perceiving and acting in the environment is a useful abstraction as it provides a black-box view on the agent architecture and encapsulates dynamic aspects related to agents’ run time. Because of the richness of research and development in agent-oriented systems, there exist a large number of different agent architectures and agent platforms each having unique system run-time. As it is stated in [16], an agent platform is a software environment that provides recourses and functionalities for software agents to operate. Some of these recourses and functionalities are[21]:

- Sophisticated communication infrastructures (e.g. in JADE, FIPA-OS),
- Support for agent mobility (e.g. in ADK, Voyager).
- Reasoning engine(e.g. in Jack, ABLE)

An agent platform may also provide a special platform language for manipulating agents interfaced with object oriented language like Java or C++. The second method is to implement agents from scratch by using an existing object oriented programming language with agent-built support libraries, well, this method is not so simple as it sounds, first of all we need to bridge the differences of agent and objects to be managed in the same programming paradigm level of abstraction (run time, programming languages). As we noticed from section 2.3, 2.4, the bottom line that differentiate object and agents is agent has internal state, environment, and communication protocols with other software entities (user, object, other agent).

---

* There are methodologies and modelling techniques published for AOSE (e.g. AUML [13], Gaia [14], MaSE [15]…etc), but are too immature to be used in real agent software application.
these characteristics are more likely as knowledge base representation and manipulation with given rules, but objects are more concern with attribute and methods represented with absolute data. The existing standard object-oriented languages cannot be used to support the implement agent components [12]. The proposed libraries for agents are defined as knowledge containers that can be used by agents to represent their mentality and behavior, with a thread of interaction for each agent can be created using threading in object oriented language this thread is created as soon as an agents starts to operate (agent born) it will constantly running as memory resident entity (agent life) and its operation (role) cannot be disturbed or halted until a superior interrupt occurs (agent death).

The first method helps us to develop software based on “agent only” architecture, and application which gives more emphasis on agency theory, an appropriate kind of architecture for this method is application based on MAS and ABS modeling (see next section). But using the first method in a general software application like banking, accounting …etc, is not a good idea basically because the developers will fall into restriction rather than a help [16]. The most severe restriction is that the developer must “agentify” all software components to be manipulated with the same manner. The second method is more appropriate for general software kind applications that have objects and agents cooperate to perform the main tasks of software application, in this method the prospective AOSE would be the suitable engineering process for this kind of software development, which has object (non-agent) and agent entities.

In any engineering process the greatest burden and priority is given for the building tool. For example, if an architect shows amazing designs for a house on the water service, this is useless until tools and equipment for building such house are provided. The same story applies with software engineering and programming tools. Therefore, any software engineering methodology to be useful, it must be supported by appropriate programming tools. Most general software systems may contain objects as well as agents, and objects do not have to be more than what they really are. On the other hand, agents will loose most of their characteristics if they are going to be treated as special kind of objects. We conjecture that in the most common and generic software systems we have to deal with both “objects” and “agents”. Therefore, we concern with agent-object coexistence. AOSE should have the ability to represent agents as well as objects in the system. Reflectively, we have to build the most suitable implementation tool (programming language) that has this kind of ability, which is an object-oriented language (using C++, or Java) with agent-oriented support libraries.

4.0 AOSE Evolution and Definition

The purpose of AOSE is to incorporate agents into software systems. This research field is relatively new and currently on the working table, and has been motivated by at least two major research fields like software engineering and artificial intelligence. Therefore, it is naturally to have very closely related research topics like:

- Agent Oriented Software Engineering (AOSE),
- Agent –Base System (ABS) modeling,
- Multi-Agent System (MAS) modeling.

To clarify the scope of these overlapping topics, let’s go back step when object oriented software engineering (OOSE) came up. The intention was to model software applications using the object-oriented (OO) paradigm. According to the OO approach each entity is represented by an object. As a result, OO software contains only objects that have their own methods and attributes, which are manipulated by the programmer. Generally speaking, OO approach can be used for building any software application. Doing so, any entity could be “objectified”. For instance, a piece of information or a document can be considered as an object. On the other hand, we cannot suggest a similar development (i.e. agentification) to occur in Agent Oriented Software Engineering (AOSE), because the most common software applications and real life applications contain “agent” and “non-agent” entities.

On the other hand, as it is defined in [17] the technology for multi-agent system (MAS) stems from distributed artificial intelligence (DAI) research field. MAS and ABS modeling provide modeling techniques for software applications from the agent’s perspective. As it is already indicated, entities in such systems are “only agents” that interact/coordinate to accomplish specific goals. Therefore, agent platforms built within this context are the most appropriate to be used as implementation tools for MAS and ABS modeling techniques.

<table>
<thead>
<tr>
<th>AOSE</th>
<th>ABS &amp; MAS Modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities</td>
<td>“Agent”, “non-agent”</td>
</tr>
<tr>
<td>Programming Language</td>
<td>OO Standard language with Agent-support libraries</td>
</tr>
<tr>
<td>Software Application</td>
<td>Common, generic and traditional (with agents added)</td>
</tr>
</tbody>
</table>

Table 3: AOSE vs. MAS & ABS Modeling

Candidate areas for applying MAS & ABS are [18]:

- **Problem solving**: concerns all situations in which a software agent accomplishes tasks which are of use to human beings
- **Multi-agent simulation**: Frequent use of simulation is used to try to explain or forecast natural phenomena.
- **Building artifice worlds**: Constructing of synthetic world for understanding the influence of behaviour on a regulation society (virtual world)
- **Collective robotics**: Creations of not only a single robot, but an assembly of robots, which cooperate to accomplish a mission. evolving through interaction, adaptation and reproduction of relatively
• **Program design:** Design a computer software which go beyond the data processing techniques but capable of autonomous agents functioning in physical distributed universe.

Alternatively, AOSE aims to merge agency characteristics into software engineering. This new approach must comply with software engineering criteria and agent’s characteristics modeling criteria, as it is mentioned in [19]. A programming tool that complies with AOSE approach can be an OO language that provides agent-building support libraries to manipulate agent and object entities, so as to preserve their distinctive characteristics at the programming paradigm levels of abstraction (Figure 1).

Summarizing the above, the new software engineering generation AOSE is the natural development of OOSE, Table 3 show the differences between AOSE and MAS, ABS modeling, in terms of entities’ kind within the modeled system, the implementation tool (programming languages) and the nature of software application.

5.0 **Concluding Remarks (Challenges).**

Agent Oriented Software Engineering (AOSE) approaches aim to provide modeling languages and tools that systematically divide complexity into a collection of inter-related models [20]. The main challenges concern:

- Definition of a methodology, and modeling technique that represents objects as well as agents with their characteristics
- Definition of an implementation tool (programming paradigm) that is suitable for building a system where objects’ and agents’ distinctive characteristics are preserved at the programming paradigm levels of abstraction.

Concerning the second challenge, to reach agents and objects coexistence into software systems the programmer must be given the ability to manipulate agents in the same manner of manipulating objects. Agents at the runtime level have a different attitude from objects. Therefore, this should be reflected at the programming language level of abstraction in order to produce agent tool-kits or agent support libraries extending standard OO languages.

For the first challenge, to move for the next generation of software engineering, it is critical to have a clear view of what AOSE is about, and differentiate it from MAS and ABS modeling. We need to find the most suitable methodology and modeling technique to integrate agents into the right place of software systems. This modeling must represent agents with their characteristics and objects with their own attributes, without the need for “agentifying” objects or “objectifying” agents.

**References:**


[8] Charles Petrie Agent-Based Software Engineering, Stanford Networking Research Center 350 Sera Mall Stanford, CA 94305-2232 petrie@stanford.edu


[19] Federico Bergenti, Omm Shohery , Franco Zambonelli : Agent-Oriented Software Engineering EAASSS Italy 2002
