COMPUTATIONAL SCIENCE AND TECHNIQUES
Volume 5, Number 1, 2017, 541-559
© Klaipėda University, 2017

Publisher: Klaipėda University
http://journals.ku.lt/index.php/CST
Online ISSN: 2029-9966
DOI: 10.15181/csat.v5i1.1361

VARIATIONS ON AGENT-ORIENTED PROGRAMMING

Dalia Baziukė, Natalija Juščenko
Department of Computer Science and Statistics, Klaipeda University, Lithuania

Abstract. Occurrence of the agent paradigm and its further applications have stimulated the emergence of new concepts and methodologies in computer science. Today terms like multi-agent system, agent-oriented methodology, and agent-oriented programming (AOP) are widely used. The aim of this paper is to clarify the validity of usage of the terms AOP and AOP language. This is disclosed in two phases of an analysis process. Determining to which concepts, terms like agent, programming, object-oriented analysis and design, object-oriented programming, and agent-oriented analysis and design correspond is accomplished in the first phase. Analysis of several known agent system engineering methodologies in terms of key concepts used, final resulting artifacts, and their relationship with known programming paradigms and modern tools for agent system development is performed in the second phase. The research shows that in the final phase of agent system design and in the coding stage, the main artifact is an object, defined according to the rules of the object-oriented paradigm. Hence, we conclude that the computing society still does not have AOP owing to the lack of an AOP language. Thus, the term AOP is very often incorrectly assigned to agent system development frameworks that in all cases, transform agents into objects.

Keywords: agent, term, concept, methodology

Introduction

During the last decade the idea of agent-oriented (AO) software engineering has been exploited intensively. The appearance of new methodologies, tools, and concepts has led to the emergence of new terms like agent, AO programming, agent-oriented programming (AOP) language, and so on. An analysis of scientific publications, educational literature, and technical papers in the AO research area shows that scientific publications offer contradicting definitions of terms such as AOP and AOP language. For example, on the one hand, there are several works making claims about AOP languages (Shoham, 1993; Bordini et al., 2005; 2009; Wang, Chan, 2000; Boissier et al., 2011), whereas, on the other hand, there is a list of works stating that this type of language does not exist (Akbari, 2010; Juneidi, 2004). In the book by Bordini et al. (2005:113) the authors state that "the CLAIM language is supported by a dedicated platform, called SyMPA <...>, implemented in Java...". Yet, a few lines below this, on the same page, a contradicting claim is made that "the main difference of SyMPA with respect to other mobile agents platforms is that it supports agents implemented in CLAIM, an agent-oriented programming language while the other platforms support agents implemented using mainly object-oriented languages (e.g. Java in most cases)." Such misinterpretation of terms confuses the readers, causes misunderstanding of research results, and complicates application of accumulated knowledge to further research in the area. Thus, the aim of this paper is to clarify the validity of using the terms AOP and AOP language. To fulfill this aim an analysis process consisting of two phases is carried out. The following three sections cover the first phase of the analysis, which is devoted to determining to which concepts terms like agent, programming, object-oriented (OO) analysis and design, object-oriented programming (OOP), and AO analysis and design correspond. Section 3 focuses on the concept of an agent and discusses normal agents and intelligent agents. Section 4 compares the concepts of agent, object, and expert system and presents a chronology of OO languages, OO software engineering (OOSE) methodologies, AO software engineering (AOSE) methodologies, and AO development tools. Section 5 covers the second phase of the analysis process, in which an analysis of several known agent system engineering methodologies in terms of key concepts used, final resulting artifacts, and their relationship with known programming paradigms and modern tools for agent system development is carried out. Finally, Section 6 presents our concluding remarks.

1. Main concepts discussed and their interrelation

To deal with complex algorithms systematically or to develop complex software, a suitable *programming language* is needed. According to Iverson, who is the creator and developer of the APL language, "... such a programming language should be concise, precise, consistent over a wide area of application, mnemonic, and economical of symbols; it should exhibit clearly the constraints on the sequence in which operations are performed; and it should permit the description of a process to be independent of the particular representation chosen for the data" (Iverson, 1967). APL is the base for many modern programming languages. Since the early 1960s, several programming paradigms have emerged. A *programming paradigm* is assumed to be a specific style of *computer programming* (Fig. 1), which is definitely concerned with designing, writing, debugging, and maintaining the source code of computer programs. Four fundamental paradigms are known: functional, procedural, logical, and object-oriented (Gabbrielli and Martini, 2010; Goel, 2010). Each programming paradigm is expressed by one or more programming languages defined by their own strict notation. *Notation* is the grammar by which the syntax of a particular programming language is set. In modern software engineering, a grammar is usually expressed as a set of graphical elements that are even exploited in the early stages of system analysis and design. Moreover, some authors (Bergenti at al., 2004; Isern et al., 2011) treat programming languages as obligatory tools to support a particular software engineering methodology. The concepts of objects and OO methodology can be mentioned as an example. In the early 1960s the idea of OOP emerged. Kay, the pioneer of the OOP paradigm, declared that at that time there was no programming language that supported the OO paradigm, and that his initial idea of an OO language included only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things. The Simula programming language designed by Dahl and Nygaard appeared to be the first OOP language (Fig. 2) and had a strong influence on the rest of the OOP languages. Now, after fifty years there are several programming languages that use objects as the main construct in the programming language. Thus, there is no doubt that

OOP exists. As another example, consider the concepts of agents and AO methodology. These concepts require that an AOP language should exist (Fig. 3). It appears that Shoham (1993) introduced such a language. Since then, a variety of research papers have been published in the field and much effort has been devoted to developing the so-called AOP paradigm. How productive these efforts were, and whether AOP really exists is the focus of the discussion in this paper.

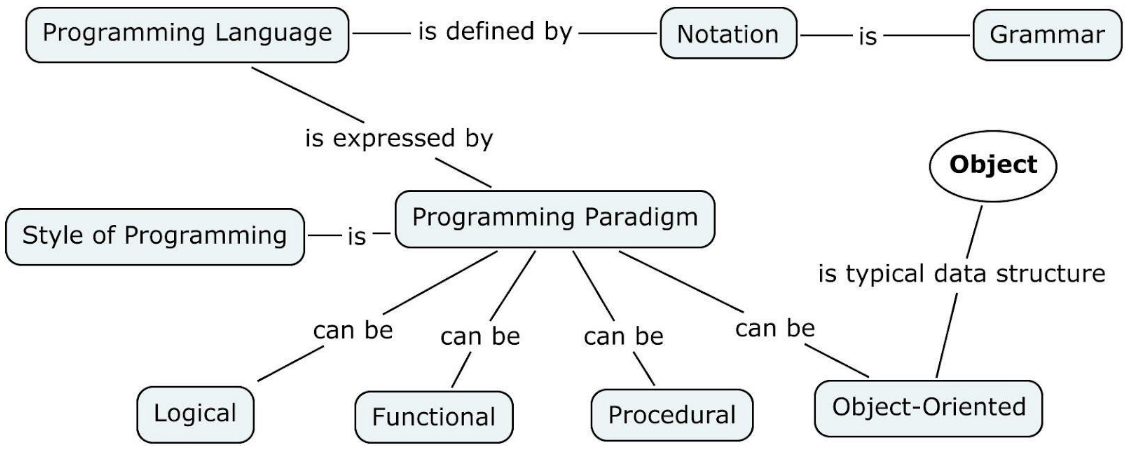

Fig. 1. **Map of main concepts discussed.**

The next section focuses on the notion of agents and discusses agent-related definitions in order to enable the distinction between the levels of abstraction that agents and objects have. This is discussed in more detail in Section 5.

3. Agent-related definitions

The topic of agents is a "hot" one in scientific literature with the discussion mainly concentrating on different understandings and definitions of the terms agent and intelligent agent. We attempt to express the point of view of some well-known scientists in the current field and present the more or less agreed-upon definitions of the terms agent and intelligent agent.

One can argue that two major viewpoints coexist in agent-based research with the *strong* one being the so-called artificial intelligence (AI) viewpoint, and the *weak* one being the software engineering (SE) viewpoint. According to the strong viewpoint (Jennings and Wooldridge, 1998; Wooldridge, 2009) an agent is proactive, intelligent, and must perform conversations instead of doing client-server computing. The weaker one states that an agent is a software component with internal (either reactive or proactive) threads of execution, and which can be engaged in complex and stateful interaction protocols. Moreover, a multi-agent system can be viewed from the AI or SE perspective. On one hand, a multi-agent system is defined as a society of individual (AI software) agents that interact by exchanging knowledge and by negotiating with each other to achieve either their own interest or some global goal. On the other hand, it can be viewed as a software system made up of multiple independent and encapsulated loci of control (i.e., agents) interacting with each other in a specific application context (Shehory and Sturm, 2004).

The metaphor of an agent is very convenient as a design view, where agents provide a "natural and elegant means to manage complexity" (Luck et al., 2003). The complexity of many systems arises from the interactions between components of the system, and an agent paradigm provides a natural way of modeling such interactions (Boman and Holm, 2004; Luck et al., 2003). Thus, agents can be viewed as a design metaphor used to design software systems with components that interact with each other and abstractly may be understood as individual goal seeking items. Jennings (2000) states that agents provide designers and developers with a way of structuring an application around autonomous, communicative elements, leading to the construction of software tools and infrastructure to support the design metaphor.

Thus, software agents are referred to as a *design metaphor* to ensure that a complex system has the desired behavior with the components expressing certain delegated tasks on the one hand, and on the other, as *autonomous components* that support effective behavior of a system in a dynamic and open environment.

3.1. Normal Agents

According to Norvig and Russell (2010), an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. Franklin and Graesser (1996) defined an agent as a system situated within and part of an environment that senses the environment and acts on it over time, in pursuit of its own agenda and so as to effect what it senses in the future. According to Wooldridge and Jennings (1995) an agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives. All three definitions agree on two things. First, an agent is something situated in some environment and interacting with it, and second, it is able to perform autonomous action. *Autonomy is the main factor that characterizes agency.*

3.2. Intelligent Agents

When does an agent become intelligent? Similar to the notion of an agent, it is hard to find a single exact definition strictly describing the term "intelligent agent". For example, in (Sycara et al., 1996) an agent is considered to be intelligent if it is taskable[1], network-centric[2], semi-autonomous, persistent[3], trustworthy to users, anticipatory[4], active[5], collaborative[6], flexible[7], and adaptive[8]. The list of properties for an intelligent agent is very ambitious and none of the existing agent systems meet these criteria completely. In (Jennings and Wooldridge, 1998; Wooldridge, 2009) an intelligent agent is described as an

[1] Able to take direction from humans and other agents.

[2] Distributed and self-organising, mobility may be desirable in some cases.

[3] Capable of unattended operation for long periods.

[4] Anticipate user's needs for information using various user models.

[5] Can initiate actions where relevant.

[6] Able to interact with humans and other agents.

[7] Can handle heterogeneity of agents and information resources.

[8] Adjust to changing user needs and task environments.

agent having the capabilities of reactivity[9], proactiveness[10], and social ability[11]. In addition, intelligent agents have properties ascribed to “normal” agents.

3.3. Agent Environment

An agent acts in its environment, which is usually described using the notion of property. The most exploited example of environmental states in the literature is cold, warm, hot when discussing an agent used to implement a thermostat (Booch, 1994; Król and Nguyen, 2008; Poole and Mackworth, 2010; Wooldridge, 2009). The environment may be in any of a finite set E of discrete instantaneous states, $E = \{e, e', \ldots\}$.

For example, in the case of intelligently adapting an online system for educational purposes (Baziukaitė, 2006), the modeling has to be done around the entities of a Learner and a Teacher. Depending on this, in a very general case, two subsets of E have to be defined, one for the Teacher Agent and the other for the Learner Agent:

$$E = E_T \cup E_L, , \qquad (1)$$

where $E_T = \{e_T, e'_T, \ldots\}$ and $E_L = \{e_L, e'_L, \ldots\}$ are discrete sets of finite instantaneous states for environments E_T and E_L, respectively.

To achieve the final state, the goal state, the agent has to perform a *run*. During the run, the agent takes certain *actions* that transform the agent‘s current state into the next one defined in the sequence (Baziukaitė, 2007b; Wooldridge, 2009). Usually, a run, r, of an agent in an environment is a sequence of interleaved environment states and actions

$$r : e_1 \longrightarrow^{\alpha_1} e_2 \longrightarrow^{\alpha_2} \ldots \longrightarrow^{\alpha_{n-1}} e_n, , \qquad (2)$$

where e_i, $i = 1 \ldots n$ are the finite states in the environment E and α_i, $i = 1 \ldots n$ are the actions that transform a state in the environment.

The next section compares the concepts of agent, object, and expert system and presents a chronology of OO languages, OOSE methodologies, AOSE methodologies, and AO development tools. This is done to ascertain development tendencies of AO notation and determine in which software engineering steps the concept of agent is currently being applied.

4. Agents, Objects, and Expert Systems

Are there any distinctions between agents and objects, and also between agents and expert systems that can be or should be highlighted? While dealing with various sources and applying the agent-based approach as a design metaphor, one can certainly stress that an agent is not the same as an object, and an agent is not the same as an expert system. According to (Booch, 1994; Booch et al., 2007; Jacobson, 1992) objects are used in

[9] Agents are able to perceive their environment and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.

[10] Agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives.

[11] Agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

software development to implement abstract data structures. An object is an element that has a state and a number of operations (behavior) to either examine or change its state. In the coding stage, OOP languages provide extensive syntactic and semantic support for object handling. An object is the foundation of OOP, and is a fundamental data type in OOP languages.

The main features distinguishing an agent from an object (Petrie, 2001; Shoham, 1993; Wooldridge, 2009) are the embodiment of a stronger notion of autonomy than that which objects have; the ability to decide for themselves whether to perform an action on request from another agent; and the capability of flexible (reactive, proactive) behavior, about which the standard object model has nothing to say. A multi-agent system is inherently multi-threaded in that each agent is assumed to have at least one thread of control.

Similarly, one could argue that agents are not the same as expert systems (Wooldridge, 2009) because classic expert systems (Jackson, 1998; Russell and Norvig, 2010) are not coupled to any environment in which they act, but instead act through users as the "middleman"; they are not capable of reactive or proactive behavior; and are not generally equipped with social ability in the sense of cooperation, coordination, and negotiation. Features of intelligent agents are really attractive, with some features, like proactiveness, social ability, and so on, making them capable of coping with a higher level of abstraction than objects.

Intelligent agents have found their place in different types of applications: industrial applications (process control, manufacturing, and air traffic control), commercial applications (Baig, 2012) (information management, business process management, and electronic commerce), medical applications (Isern et al., 2010) (patient monitoring and healthcare), and entertainment (games, interactive theatre, and cinema) (Jennings and Wooldridge, 1998). The application field for intelligent agents is being extended to educational systems, particularly through application of the paradigm in virtual learning environments, see, for example (Aylett and Luck, 2000; Baziukaitė, 2006, 2007a, 2007b; Evers and Nijholt, 2000; Hobbs, 2002; Liu and Chen, 2005, Jurado et al., 2012).

After considering the literature on AOSE, one gets the impression that the story of OOSE is being repeated. Yet, on closer inspection (Fig. 2 and Fig. 3), it is clear that the situation is different. In the OO case, real programming languages that supported objects as their main language construct were created and presented in the beginning. Simula (introduced in 1966) and Smalltalk (introduced in 1971) are examples of such languages. Moreover, some twenty years later (Fig. 2) OO modeling languages have been introduced. This possibly has a connection with the production of software, which appeared more demanding as the software systems became more complex and the task of designing such systems required tools supporting the object notation even at the software design level. The first OO modeling language was introduced in 1987 almost at the same time as commercial C++ compilers appeared on the market (Fig. 2). It took a further ten years for these OOSE methodologies to reach maturity (in 1997) and have since been standardized as the Unified Modeling Language (UML).

In the AO paradigm, the opposite situation is observed (Fig. 3). Since 1991, AO modeling languages/methodologies have been introduced, with the process of introducing new ones still ongoing. Nevertheless, in 2012 there is still no unified (at the standard level)

AO modeling language. It must be noted, that most of the known AO modeling languages make use of UML models to express the essence of their own model. Recently, some authors have mentioned the need for an AOM standard, which is seen as a key tool for AOSE to become an industrial standard in software design. But at the same time there is still no AOP language[12]. Thus, it turns out that while computer systems are becoming more and more complex, the task of designing them is becoming more and more sophisticated. This is the reason why the scientific computing society is searching for more efficient software modeling tools. Currently it seems that the concept of agents coming from the AI field has found valid application in the SE environment. The *agent* concept with a higher level of abstraction than an *object* allows reflection of the behavior of complex systems in a more natural way. However, in the final implementation stage of agents in program code, each agent is transformed to an object.

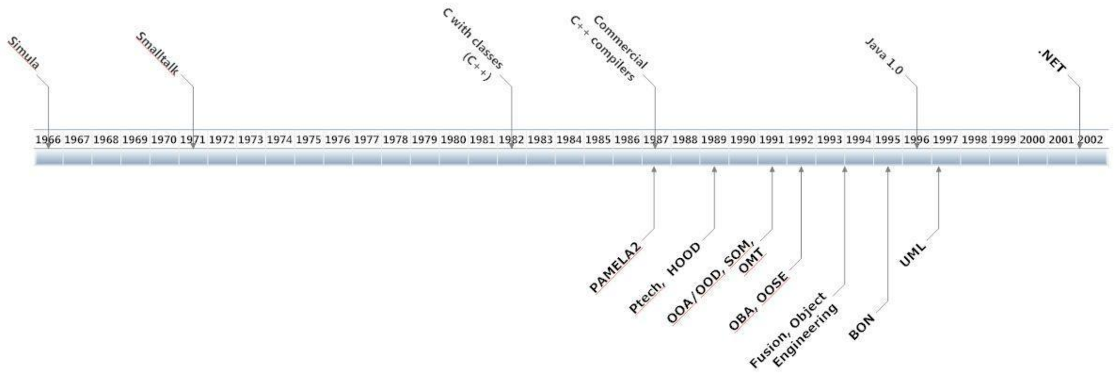

Fig. 2. **The timeline depicts the year of introduction of various OOP languages (above the timeline) and OOSE methodologies (below the timeline).**

[12] Shoham, in 1993, introduced the AGENT-0 language, with a LISP based interpreter and the main language construct being a list. Later some agent development frameworks such as JACK, JADEX, and JADE were introduced. These are Java based tools, with the main language construct being an object. These simply transform the higher level of abstraction of the agent to the lower level of an object.

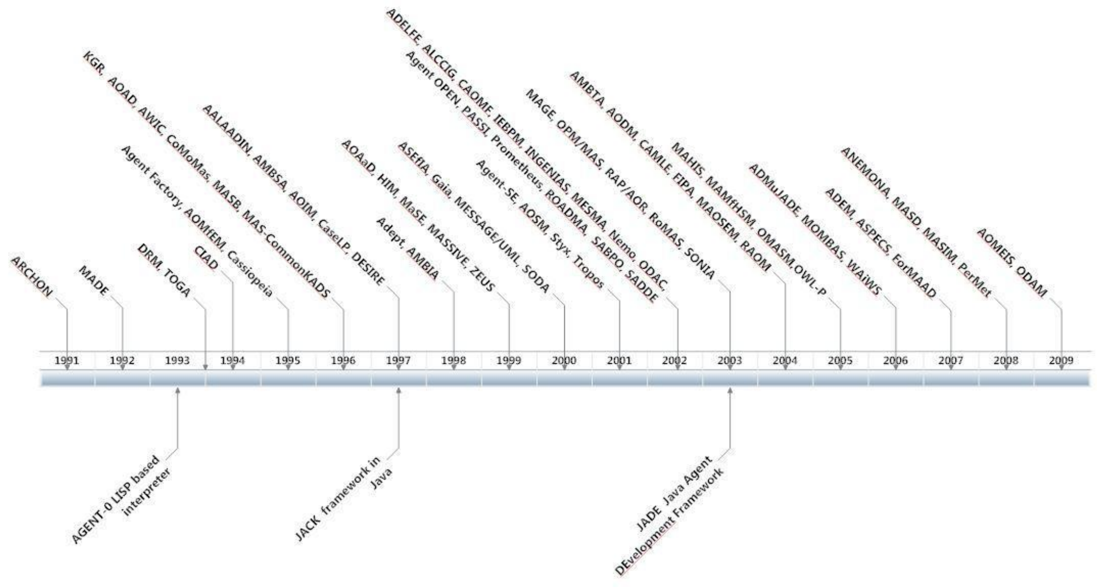
ARCHON
MADE
DRM, TOGA
CIAD
Agent Factory, AOMfEM, Cassiopeia
KGR, AOAD, AWIC, CoMoMas, MASB, MAS-CommonKADS
AALAADIN, AMBSA, AOIM, CaseLP, DESIRE
Adept, AMBIA
AOAaD, HIM, MaSE, MASSIVE, ZEUS
ASEfIA, Gaia, MESSAGE/UML, SODA
Agent-SE, AOSM, Styx, Tropos
ADELFE, ALCCIG, CAOME, IEBPM, INGENIAS, MESMA, Nemo, ODAC
Agent OPEN, PASSI, Prometheus, ROADMA, SABPO, SADDE
MAGE, OPM/MAS, RAP/AOR, RoMAS, SONIA
AMBTA, AODM, CAMLE, FIPA, MAOSEM, RAOM
MAHIS, MAMfHSM, OMASM,OWL-P
ADMuJADE, MOMBAS, WAiWS
ADEM, ASPECS, ForMAAD
ANEMONA, MASD, MASIM, PerMet
AOMEIS, ODAM
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
AGENT-0 LISP based interpreter
JACK framework in Java
JADE Java Agent DEvelopment Framework

Fig. 3. **The timeline depicts the year of introduction of various AOSE methodologies (above the timeline) and AO development frameworks (below the timeline).**

The next section is devoted to the comparative analysis of several known agent system engineering methodologies in terms of key concepts used, final resulting artifacts, and their relationship with known programming paradigms and modern tools for agent system development. This is done to understand the respective levels of abstraction for agents and objects.

5. Agent-Oriented Design

In this section a comparison of several well-known AOSE methodologies is presented. The selection of methodologies was done based on the availability of documentation describing each one, the familiarity of the agent community with it, and whether it has been developed over an extended time period, according to feedback from users other than the developers of the methodology. Another important factor was whether the methodology has an implementation into the code stage and whether there any tools developed to support the analysis, design, and implementation stages of the methodology. Based on these criteria we chose nine methodologies, Tropos, MaSE, Gaia, PASSI, Prometheus, Moise+, INGENIAS, AGR, and ASPECS (Table 1).

For example, the Tropos methodology (Mylopolous and Castro, 2000; Bresciani et al., 2004; Tropos, 2012) is intended to cover early and late requirements, architectural design and detailed design, and also to interface with agent programming platforms. It is based on a UML-type language and methodology, but includes the notion of a goal in the highest level of system design. It should be noted that UML is a standardized general-purpose modeling language in the field of OO software engineering (Booch et al., 2007). The tools supporting the Tropos methodology are Si*Tool, TAOM4E, GR-Tool, T-Tool, DW-Tool, OpenOME, DESCARTES, and SecTro. Only TAOM4E supports code generation according to JADEX (Jadex, 2012), which allows programming intelligent software agents in XML and Java and can be deployed on different kinds of middleware such as JADE

(Jade, 2012), a software framework fully implemented in the Java language. Java is a general-purpose, concurrent, class-based, OO language that is specifically designed to have as few implementation dependencies as possible (Gosling et al., 2005). Thus, it can be concluded that an object is the main artifact derived from the design process.

The Multiagent System Engineering (MaSE) methodology (DeLoach, 2004a, 2004b; DeLoach et al., 2001; DeLoach and Wood 2001) was originally developed for modeling complex agent based systems. This methodology is based on what is referred to as the Belief-Desire-Intention (BDI) paradigm, but also uses some basic models that are common to OO design methodologies. A supporting tool is agentTool (DeLoach and Wood, 2001; DeLoach, 2001; agentTool III, 2012), which is a Java-based graphical development environment.

Gaia (Woodridge et al., 2000) was specifically developed for the task of analysis and design of agent-based systems by Wooldridge, Jennings, and Kinny. This methodology is applicable to a range of multi-agent systems and is founded on the view of a multi-agent system as a computational organization consisting of various interacting roles. Gaia allows switching systematically from a statement of requirements to a design (moving from abstract to increasingly concrete concepts). It borrows some terminology and notions from OO analysis and design, but provides an agent-specific concept that helps to understand and model complex systems. The overall process of the design of a multi-agent system is divided into two stages: analysis and design. Analysis and design can be thought of as the process of developing increasingly detailed models of the system to be constructed. Methodology is supported by Jade - Java Agent DEvelopment Framework (Jade, 2012).

PASSI is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies, integrating design models and concepts from both OO software engineering and artificial intelligence approaches using the UML notation (Cossentino, 2005). As UML is an OO modeling language the main artifact from the design process in PASSI is also an object. The supporting tool, AgentFactory (Agent Factory, 2012) provides a set of supporting kits that are mainly based on OO languages.

The Prometheus methodology supports, but is not limited to, the design of BDI systems. The lowest level of design leads to the code, but this needs to be modified to the programming paradigm being targeted (Padgham and Winikoff, 2005). The tools supporting Prometheus are PDT (Padgham et al., 2008) and JACK. The latter supports code generation to Java.

The Moise+ methodology (Hübner et al., 2002) follows the organisational model for multi-agent systems based on notions like roles, groups, and missions. The supporting tool is Jason, a platform for the development of multi-agent systems developed in Java (Jason, 2012).

INGENIAS methodology (Fuentes-Fernandez et al., 2010) develops its models according to the steps of the Rational Unified Process (RUP). The INGENIAS Development Kit (IDK, 2012) facilitates the development of multi-agent systems by supporting the INGENIAS methodology. The development kit is written in Java, so the lowest level needs to be supported by an OO implementation platform.

The AGR methodology is based on the three key notions of an agent, group, and role (Ferber et al., 2003). In addition, the Gaia methodology was proposed for use in filling the roles and relating them to the general structure. The MadKit tool (Madkit, 2012) was built on the AGR organizational model. It is a modular and scalable multi-agent platform written in Java.

The ASPECS methodology provides a step-by-step guide from requirements to code allowing the modeling of a system at different levels of detail through refinement (Cossentino et al., 2010). UML is used as the base modeling language. The AO solution needs to be adapted to the chosen OO implementation platform. Supporting tools are Janus, Madkit, and Jade. Janus (Janus, 2012) is a multi-agent platform fully implemented in Java.

Table 1 gives the general steps of the nine methodologies compared and allows comparison of key concepts used to express a variety of models applied in each particular methodology. The structure of Table 1 is as follows: the first column denotes the steps that persist in each methodology. We see that two steps expressing analysis and design are common to all of them. Other rows in the table show models that constitute the analysis and design steps for all methodologies. Each model is followed by a list of key notions giving us a better insight into the level of detail achieved in creating each model, and highlighting similarities between methodologies. The last row in the table shows the relationship of the methodologies with known programming paradigms and modern tools for agent system development.

Table 1. Comparison of various AO design methodologies.

1	2	3	4	5	6	7	8	9	10
Methodology Stage	**Tropos**	**MaSE**	**Gaia**	**PASSI**	**Prometheus**	**Moise+**	**INGENIAS**	**AGR**	**ASPECS**[13]
Analysis	Early Requirements; Late Requirements	Analysis Phase	Analysis Stage	System Requirements Model; Agent Society Model	System Specification	Structural Specification; Functional Specification	-		System Requirements Analysis
	• Actor Modeling key concepts: actor • Dependency Modeling key concepts: interdependency, actor,	• Capturing Goals key concepts: system goal, requirement, goal hierarchy • Applying Use Cases key concepts:	• Roles Model key concepts: role, activity, responsibility, permission • Interactio	• Domain Description key concepts: use case, functional description • Agent Identification key concepts:	• Defining Functionality key concepts: functionality descriptor, use case scenario	• Individual Level key concepts: role, abstract role, global plan • Social Level key concepts:	• Use Case definition key concepts: use case diagram, role • Interaction Definition key concepts:	• Groups Identification key concepts: group structure diagram, role, interactions, dependenc	• Domain Requirements Description key concepts: use case, functional description • Problem

[13] Can be seen as extension of PASSI.

1	2	3	4	5	6	7	8	9	10
	goal, plan, resource	use case, sequence of events • Refining Roles key concepts: role, associated task	n model key concepts: interactio n between roles, protocol	stereotyped UML package • Role Identificati on key concepts: sequence diagram, responsibili ty, scenario • Task Specificatio n key concepts: use case, capability, agent • Domain Ontology Description key concepts: class diagram, agent interaction • Role Description key concepts: class diagram, communica tion, interagent dependency • Protocol Description key concepts: sequence diagram, communica tion protocol		role, link • Collective Level key concepts: compatibili ty constraint, group specificatio n, role • Social Scheme key concepts: global goal, global plan, goal decomposit ion, sequence, choice, parallelism, preference order	interaction diagram, interaction protocol • Organizatio n High-Level and Low-Level key concepts: organizatio n diagram • Tasks and Goals Definition key concepts: task, goal, role	y • Definition of Organizati onal Structure key concepts: *Cheeseboar d* Diagram, group, agent, role • Decompos ition of Organizati onal Structure key concepts: sub-componen t • Describin g Organizati onal activities key concepts: organizati onal sequence diagram, group, role, message, playing a role, leaving a role, creation of group, entering a group • Definition of Roles[14] key concepts: role, interactio n	Ontology Descriptio n key concepts: ontology, class diagram • Organizati on Identificat ion key concepts: role, interactio n, context, use case diagram • Interactio ns and Role Identificat ion key concepts: behavior, abstract role, class diagram • Scenario Descriptio n key concepts: sequence diagram, role, interactio n • Role Plan key concepts: activity diagram, goal (requirem ent), task • Capacity Identificat ion key concepts: class diagram, role

[14] It is suggested that the Gaia methodology be used to define roles (Ferber et al., 2003).

1	2	3	4	5	6	7	8	9	10
									behavior
Design	Architectural Design; Detailed Design	Design Phase	Design Stage	Agent Implementation Model; Code Model; Deployment Model	Architectural Design; Detailed Design	Deontic Specification	-	-	Agent Society Design; Implementation; Deployment
	• Goal Modeling key concepts: plan, resource, softgoal •Plan Modeling key concepts: UML activity diagram • Capability Modeling key concepts: individual capabilities, social capabilities • Agent Interaction Modeling key concepts: UML sequence and activity diagrams, object	• Creating Agent Classes key concepts: agent class, conversation • Constructing Conversations key concepts: coordination protocol • Assembling Agents key concepts: agent architecture • System Design key concepts: deployment, number, type and location of agent	• Agent Model key concepts: agent type, agent instance • Services model key concepts: service • Acquaintance Model key concepts: communication link	• Agent Structure Definition key concepts: class diagram • Agent Behavior Description key concepts: activity diagram, statechart • Code Reuse Library key concepts: class diagram, activity diagram, associated code • Code Completion Baseline key concepts: source code • Deployment Configuration key concepts: allocation of agents	• Agent acquaintance Diagram key concepts: agent descriptor, interaction, event • Shared Data Objects key concepts: data source, OO techniques • System Overview Diagram key concepts: agent, event, shared data object • Interaction Diagram key concepts: interaction, protocol • Refinement of Capabilities key concepts: events, interactions, data • Agent Overview Diagram key concepts: capability, event flow, task flow,	• Defining Obligations and Permissions key concepts: obligation, permission, role, mission, global goal	• Environment Definition key concepts: internal and external application, event, task • Agent Definition key concepts: role, task • Components Definition key concepts: component diagram • Deployment Definition key concepts: deployment diagram, locations, number of agents	-	• Solution Ontology Description key concepts: concept, predicate, action, class diagram • Communication Ontological Description key concepts: communication, conversation, class diagram • Role Behavior Description agent task, agent action, statechart, activity diagram • Protocol Description key concepts: interaction, communication, scenario, sequence diagram • Organisation Dependen

1	2	3	4	5	6	7	8	9	10
					plan descriptor, event descriptor, data descriptor, data dictionary				cies Description key concepts: capacity, service, class diagram • Role Constraint Identification key concepts: concurrency, scheduling, class diagram • Holarchy Design key concepts: agent, holon, holonic cheese board diagram, dynamic definition rules, holon government description • Holon Architecture Definition key concepts: class diagram, • Code Reuse key concepts: • Code Production of Organizations and Roles key concepts: code •

1	2	3	4	5	6	7	8	9	10
									Organizations and Roles Unit Tests key concepts: test cases • Code Production of Holons key concepts: holon code • Holon Unit Tests key concepts: test cases
Implementation	• JACK resulting artifacts: Java language objects	• AgentTool resulting artifacts: objects in OOP languages	• JADE resulting artifacts: Java language objects	• C++ based systems • JADE resulting artifacts: objects in OOP languages	• JACK resulting artifacts: Java language objects	• Jason resulting artifacts: Java language objects	• IDK (INGENIAS Development Kit) resulting artifacts: objects in OOP languages	• Madkit resulting artifacts: Java language objects	• Janus resulting artifacts: Java language objects

Keeping in mind the comparison above (Table 1) and referring back to Shoham's work (1993), we conclude that currently, agent is still a higher level concept than object owing to the lack of an AOP language.

6. Conclusion

Through this paper, we set out to share our thoughts on the validity of using terms like agent, AO design methodology, and AOP, while keeping an eye on OO design. The main motivation for carrying out this analysis was the numerous scientific publications presenting AOP tools, even going so far as to call them languages. The purpose of this paper was to find an answer to the question: does AOP really exist?

Analysis of the concepts (Section 1) has shown that programming is concerned with designing, writing, debugging, and maintaining the source code of computer programs. To produce source code, one of four available programming paradigms (logical, functional, procedural, and object-oriented) must be applied, supported by appropriate programming languages, in which predicates, functions, procedures or objects are implemented as the main basic units of the programming language.

Results of the agent-related definitions' analysis (Section 3) and those from comparisons of the concepts of agents and objects (Section 4) and their evolution process, allow us to conclude that agent is a concept with a higher level of abstraction than that of an object.

Comparison of the evolution process of these concepts allowed us to formulate the hypothesis that in the implementation stage the object remains the basic unit.

The comparative analysis of AO methodologies shows that one of the main goals of the implementation stage is to transform agents into objects. A variety of tools has been developed for this purpose in order to bridge agents to the OO paradigm. This implies that no true AOP language currently exists.

In recalling the main goal of this paper, it is a mistake to refer to these tools as AOP languages as they do not correspond in full to the programming concept, and should be rather called an AO development frameworks.

References

Agent Factory, 2012. Agent Factory home page, http://www.agentfactory.com/index.php/Main_Page (last access July 20, 2012).

agentTool III, 2012. agentTool III Homepage, http://agenttool.cis.ksu.edu/ (last access July 02, 2012)

Akbari, O.Z., 2010. A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance. Journal of Computer Engineering Research, 1(2), 14–28.

Aylett, R., Luck, M., 2000. Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments. Applied Artificial Intelligence 14(1), 3-32.

Baig, Z.A., 2012. Multi-agent systems for protecting critical infrastructures: A survey. Journal of Network and Computer Applications 35, 1151–1161.

Baziukaitė, D., 2006. Approach to an Adaptive and Intelligent Learning Environment. In: Elleithy, K., Sobh, T., Mahmood, A., Iskander, M., Karim, M. (Eds.), Advances in Computer, Information, and Systems Sciences, and Engineering, Proceedings of IETA 2005, TeNe 2005 and EIAE 2005. Springer, XV, pp. 399-406.

Baziukaitė, D., 2007a. Learner oriented methods to enhance capabilities of virtual learning environment, Doctoral Dissertation. Vytautas Magnus University, Kaunas.

Baziukaitė, D., 2007b. Investigation of Q-learning in the context of a virtual learning environment. Informatics in Education 6, 255–268.

Bergenti, F., Gleizes, M.P., Zambonelli, F., 2004. Methodologies And Software Engineering For Agent Systems: The Agent-oriented Software Engineering Handbook. Springer.

Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A., 2011. Multi-agent oriented programming with JaCaMo. Science of Computer Programming, doi:10.1016/j.scico.2011.10.004.

Boman, M., Holm, E., 2004. Multi-agent systems, time geography, and microsimulations. In: Olsson, M.O., Sjstedt, G. (Eds.), Systems Approaches and their Application, Chapter 4. Kluwer Academic, pp. 95-118.

Booch, G., Maksimchuk, R.A., Engle, M.W., Young, B.J., Conallen, J., Houston, K.A., 2007. Object-Oriented Analysis and Design with Applications, 3rd ed. Addison Wesley Professional.

Booch, G., 1994. Object-Oriented Analysis and Design with applications. Addison-Wesley Pub.Co., NewYork.

Bordini, R.H.; Dastani, J. M.; Dix, A.E. Fallah-Seghrouchni (Eds.), 2005. Multi-Agent Programming: Languages, Platforms and Applications Vol. I, Springer, 2005.

Bordini, R.H.; Dastani, M. J.; Dix, A.E. Fallah-Seghrouchni (Eds.), 2009. Multi-Agent Programming: Languages, Tools and Applications Vol. II, Springer, 2009.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A., 2004. TROPOS: an agent-oriented software development methodology. Auton. Agents Multi-Agent Syst. 8, 203–236.

Cohen, P.R., Levesque, H.J., 1990. Intentions Choice with Commitment. Artificial Intelligence 42, 213–261.

Cossentino, M. 2005. From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies. Idea Group Inc.

Cossentino, M., Gaud, N.,· Hilaire ·V., Galland, S., · Koukam, A., 2010. Aspecs: an Agent-Oriented Software Process for Engineering Complex Systems - How to Design Agent Societies under a Holonic Perspective. Journal for Autonomous Agents and Multi-Agent Systems (JAAMAS) 20(2), 260-304.

DeLoach, S.A., 2001. Analysis and Design using MaSE and agentTool. In: Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference, Oxford OH, March 31 - April 1, 2001. pp. 1-7.

DeLoach, S.A., 2004a. The MaSE methodology. In: Bergenti, F., Gleizes, M.P., Zambonelli, F. (Eds.), Methodologies and Software Engineering for Agent Systems: The Agent-oriented Software Engineering Handbook. Kluwer Academic Publishers, pp. 107-125.

DeLoach, S.A., 2004b. The MaSE methodology. Methodologies and Software Engineering for Agent Systems. Multiagent Systems, Artificial Societies, and Simulated Organizations 11, Part II, 107-125.

DeLoach, S.A., Wood, M.F., Sparkman, C.H., 2001. Multiagent system engineering. International Journal of Software Engineering and Knowledge Engineering 11(3), 231–258.

DeLoach, S.A., Wood, M.F., 2001. Developing Multiagent Systems with agentTool. The Seventh International Workshop on AgentTheories, Architectures, and Languages (ATAL-2000), July7-9.

Dignum, V., 2004. A Model for Organizational Interaction: Based on Agents, Founded in Logic. University of Utrech, Utrech, The Netherlands.

Evers, M., Nijholt, A., 2000. Jacob-Ananimated instruction agent in virtual reality. In: Advances in Multimodal Interfaces - ICMI 2000, Proc. Third International Conference on Multimodal Interfaces, 526–533.

Franklin, S., Graesser, A., 1996. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag, UK, pp. 21-35.

Ferber, J., Gutknecht, O., Michel, F., 2003. From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Müller, J., Odell, J. (Eds.), 4th International Workshop on Agent-oriented Software Engineering IV, AOSE 2003. Springer Berlin/Heidelberg, Melbourne, Australia, p. 214–230.

Fuentes-Fernández, R., García-Magariño, I., Gómez-Rodríguez, A.M., González-Moreno, J.C., 2010. A technique for defining agent-oriented engineering processes with tool support. Eng. Appl. Artif. Intell. 23, 32–444.

Gabbrielli, M., Martini, S., 2010. Programming Languages: Principles and Paradigms. Springer-Verlag, London.

Goel, A., 2010. Computer Fundamentals. Pearson Education.

IDK, 2012. IDK home page, http://grasia.fdi.ucm.es/main/?q=en/node/127 (last access July 20, 2012)

Iverson, K.E., 1967. A Programming Language. John Wiley and Sons, Inc.

Gosling, J., Joy, B., Steele, G., Bracha, G., 2005. The Java Language Specification, 3rd ed. Addison-Wesley.

Hobbs, D.L., 2002. A Constructivist Approach to Web Course Design: A Review of the Literature, International Journal on E-Learning, April-June, 60-65.

Hübner, J.F., Sichman, J.S., Boissier, O., 2002. MOISE+: towards a structural, functional, and deontic model for MAS organization. In: Castelfranchi, C., Johnson, W.L. (Eds.), First International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2002. ACM Press, Bologna, Italy, pp. 501–502.

Isern, D., Sánchez, D., Moreno, A., 2010. Agents applied in health care: a review. International Journal of Medical Informatics 79, 145-166.

Isern, D., Sánchez, D., Moreno, A., 2011. Organizational structures supported by agent-oriented methodologies. The Journal of Systems and Software 84, 169-184.

Jackson, P., 1998. Introduction To Expert Systems. Addison Wesley.

Jacobson, I., 1992. Object-Oriented Software Engineering. Addison-Wesley.

Jade, 2012. Jade project home page, http://jade.tilab.com/, (last access July 20, 2012).

Jadex, 2012. Jadex project home page, http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview, (last access July 20, 2012).

Janus, 2012. Janus project homepage, http://www.janus-project.org/Home, (last access November 13, 2012).

Jason, 2012. Jason home page, http://jason.sourceforge.net/wp/, (last access July 23, 2012).

Jennings, N.R., 2000. On Agent-Based Software Engineering. Artificial Intelligence 117(2), 277–296.

Jennings, N.R., Wooldridge, M.J., 1998. Applications of Intelligent Agents. In: Jennings, N.R., Wooldridge, M.J. (Eds.), Agent Technology: Foundations, Applications, and Markets, 1998. Springer, pp. 3-28.

Juneidi, S.J., 2004. Toward programmining paradigms for agent oriented software engineering. In: Proc. IASTED Conf. on Software Engineering, 2004, p.428-432.

Jurado, F., Redondo, M. A., and Ortega, M., 2012. Blackboard Architecture to Integrate Components and Agents in Heterogeneous Distributed eLearning Systems: An Application to Learning to Program Journal of Systems and Software, 85, p. 1621-1636

Krol, D., Nguyen N.T., 2008. Intelligence Integration in Distributed Knowledge Management. IGI Global.

Liu, Z., Chen, B., 2005. Model and Implement an Agent Oriented E-Learning System. In: Proceedings of the 2005 International Conference on Computational Intelligence for

Modelling, Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, CIMCA-IAWTIC'05, pp. 859-864.

Luck, M., McBurney, P., Preist, C., 2003. Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent-based Computing. AgentLink, England.

Madkit, 2012. Madkit homepage, http://www.madkit.org/ (last access July 20, 2012)

Mylopoulos, J., Castro, J., 2000. Tropos: A Framework for Requirements-Driven Software Development. In: Brinkkemper, J. Solvberg, A. (Eds.), Information Systems Engineering: State of the Art and Research Themes, Lecture Notes in Computer Science. Springer-Verlag, pp. 1-12.

Padgham, L., Winikoff, M., 2005. Prometheus: A Practical Agent-oriented Methodology. In: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies. Idea Group Inc.

Padgham, L., Thangarajah, J., Winikoff, M., 2008. Prometheus Design Tool, (System Demonstration), Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008), Chicago, Illinois, USA.

Petrie, Ch., 2001. Agent-Based Software Engineering. Lecture Notes in AI, Springer-Verlag, 58-76.

Poole, D.L., Mackworth, A.K., 2010. Artificial Intelligence: foundations of computational agents. Cambridge University Press, New Yourk, USA.

Russell, S., Norvig, P., 2010. Artificial Intelligence: A Modern Approach, 3rd Edition. Prentice Hall.

Shehory, O., Sturm, A., 2004. Agent-oriented Software engineering Methodologies. In: AgentLink: 6th European Agent Systems Summer School, 5-9 July. Liverpool, UK.

Shoham, Y., 1993. Agent-oriented programming. Artificial Intelligence 60, 51-92.

Sycara, K., Pannu, A., Willamson, M., Dajun Zeng, Decker, K. , 1996. Distributed Intelligent Agents. IEEE Expert 11, pp. 36-46.

Tropos, 2012. Trotos home page, http://www.troposproject.org (last access July 20, 2012)

Wang, W.-Ch., Chan, T.-W., 2000. CAROL5: An Agent-Oriented Programming Language for Developing Social Learning Systems. International Journal of Artificial Intelligence in Education 11, 1-32.

Wooldridge, M., Jennings, N.R., Kinny, D., 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3(3), 205-312.

Wooldridge, M., Jennings, N., 1995. Intelligent Agents: Theory and practice. The Knowledge Engineering Review 10(12), 115-152.

Wooldridge, M., 2009. An introduction to Multi Agent systems, 2nd ed. John Wiley & Sons Ltd.

Dr. Dalia Baziukė received her Master's degree in Mathematics (numerical analysis and systems) from Klaipėda University in 2002 and a Doctoral degree in Informatics from Vytautas Magnus University (Kaunas) and the Institute of Mathematics and Informatics (Vilnius) in 2007. She is currently an Associate Professor in the Department of Computer Science, Klaipėda University and Director of the Virtual Learning Centre at Klaipėda University, conducting work flow and activities related to distance and e-learning, and participating in the formation of policies based on new developments establishing and supporting flexible study forms. Her research focuses on adaptivity, intelligence, and decision making processes in virtual learning environments, machine learning algorithms with various applications, and data mining.

Dr. Natalija Juščenko received a Master of Science degree in systems analysis from the Klaipėda University in 1998 and a PhD in Computer Science from the Vytautas Magnus University (Kaunas) and Mathematics and Informatics Institute (Vilnius) in 2007. She is currently working as a Lecturer in the Computer Science Department of Klaipėda University and an Engineer in the Virtual Learning Centre at Klaipėda University. Her research interests include topics related to software engineering with specific focus on database systems analysis and design, and virtual learning methodologies.