International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 3 | ISSN : 2456-3307

Selecting a methodology in Multi-Agent Systems – A Practical and Quasi-Technical Analysis of Agent-based, Object Oriented and Knowledge Engineering-based methodologies.

K. Mugoye, S. Ruoro

Department of Computer Science, Maseno University, Private Bag, Maseno, Kenya

ABSTRACT

As recognition of agents' technology registers steady improvement over years, there is an emergent need for practical methods for developing agent applications. Agent-Oriented Software Engineering (AOSE) methodologies were proposed to develop complex distributed system grounded upon the agent paradigm. Initially, the challenge was the lack of mature development methodologies for agent-based systems, efforts in the right direction to address the problem resulted in the proliferation of methodologies, which presents a new challenge that is, practitioners are challenged in that they need to select a methodology from a large number of existing methodologies. The literature in this paper suggests a necessity to the understanding of the classification of AOSE methodologies. We advocate for a view that is in twofold, first, practitioners need to first understand in a wide sense the categories of AOSE methodologies, so as to correctly link it to their intended agent solution, secondly, identify a methodology considering the availability of support features such as maturity, availability of documentation and support tools.

Keywords: Multi-Agent System, Agent Based Methodologies, Artificial Intelligence

I. INTRODUCTION

The previous two decades has seen a lot of improvement in intelligent software agent research. Originally, single agent then sophisticated single agent in complex environments to multi-agent system (MAS) organizations. Extensive research and improvements in this field over time has enabled today's agents to perform a wide variety of human-like tasks such as learning, reasoning, negotiating, self-organizing and trusting each other, just to mention.

It is quite unfortunate that very few practical MAS systems have been deployed after such an extensive period of intensive research and development. Researchers in Artificial Intelligence admit to this fact; It was noted only 12 years ago that "One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems" [3].

Scholars seem to be in consensus that analysis and design of agent-based systems requires an agent-oriented software engineering (AOSE) methodology (e.g. [3]). Efforts in addressing the problem of lack of mature software development methodologies seem to bear fruits and to date, the field has progressed considerably, resulting in the presence of many mature AOSE methodologies [4, 5] including MaSE [6] (and its successor O-MaSE [7]), Tropos [8], Gaia [9], Prometheus [10], INGENIAS [11, 12], ADEM (and its modeling language AML [13]), and PASSI [15]. Indeed, the proliferation of methodologies offers rich resources for developers to draw on, but can also be a hindrance to progress if their commonalities, divergences, and application are not readily understood. Practitioners are challenged in that they need to select a methodology from a large number of existing methodologies. This has motivated work on comparisons of methodologies; with a detailed discussion of existing methodologies given in section 2 where related research is examined. We argue that practitioners need to first understand in a wide sense

the categories of AOSE methodologies, so as to correctly link it to the solution their problem require, and second identify a methodology considering the availability of support features e.g. maturity, Availability of documentation and tools.

This paper seeks to provide insights to practitioners on how to arrive at a methodology. It presents a comparison of three methodologies, Gaia [9], PASSI [15] and MAS-CommonKADS [30]. Each methodology is drawn from a category based on the approach it adopts as agent-based, object-oriented, and knowledge engineering-based, shown in section 2. The choice of these methodologies was informed by considering maturity, availability of documentation, and availability of tool support for each category, except in cases where only one methodology is present in a category.

II. CLASSIFICATION OF AGENT ORIENTED METHODOLOGIES

Agent-oriented methodologies have several roots. They are classified according to the approach or discipline upon which they are based. A common property of these methodologies is that they are developed based on the approach of extending existing methodologies to include the relevant aspects of agents. They are broadly classified into three categories: agent-based methodologies, object oriented methodologies and their extensions, and knowledge engineering-based methodologies [4]. Figure 1 illustrates the classifications of agent-oriented methodologies.

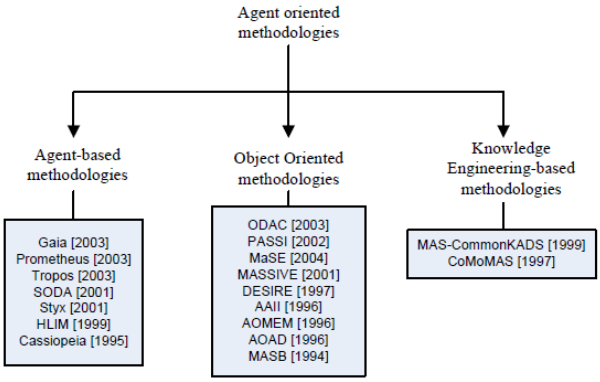

Figure 1. Clasification of agent-Oriented methodologies [4]

A. Agent-based methodologies

There are several methodologies that belong to this category such as: Gaia [9], HLIM [16], Tropos [1], Prometheus [18], SODA [19], Styx [20], and Cassiopeia [21]. The developers of such methodologies urge that the agent concept should be established without dependency on other traditional methodologies, such as object-oriented methodologies. The main reason is the inherent differences between the two entities; agents, and objects. This is because agents have a higher level of abstraction than objects. Object-oriented approaches cannot offer the same properties as agents do. They also fail to properly capture the autonomous behavior of agents, interactions between agents, and organizational structures [20]. In fact, the notions of autonomy, flexibility, and proactiveness can hardly be found in traditional object-oriented approaches [34]. As a result, object-oriented methodologies generally do not provide techniques to model the intelligent behaviour of agents [33]. Therefore, there need to be software engineering methodologies, which are specially tailored to the development of agent-based systems.

B. Object oriented-based methodologies

(Extensions of object-oriented methodologies): These methodologies belong to a category, which either extend existing object-oriented methodologies or adapt them to the aim of agent-oriented software engineering. The examples of such methodologies are ODAC [22], MaSE [23], MASSIVE [24], DESIRE [25], AAII [26], AOMEM [27], AOAD [28] and MASB [29]. Some researchers present several reasons for following this approach. Firstly, the agent-oriented methodologies, which extend the object-oriented approach, can benefit from the similarities between agents and objects. Secondly, they can capitalize on the popularity and maturity of object-oriented methodologies. In fact, there is a high chance that they can be learnt and accepted more easily. Finally, several techniques such as use cases and class responsibilities card (CRC), which are used for object identification can be used for agents with a similar purpose (i.e. agent identification) [31].

C. Knowledge Engineering-based methodologies

(Extensions of Knowledge Engineering (KE) techniques): There are, however, some aspects of agents that are not addressed in object-oriented methodologies. For instance, object-oriented methodologies do not define techniques for modeling the mental states of agents. In addition, the social relationship between agents can hardly be captured

using object oriented methodologies. These are the arguments for adopting KE methodologies for agent-oriented software engineering. They are suitable for modeling agent knowledge because the process of capturing knowledge is addressed by many KE methodologies [30]. Additionally, existing techniques and models in KE such as ontology libraries, and problem-solving method libraries can be reused in agent-oriented methodologies. Examples of such methodologies are MASCommonKADS [30] and CoMoMAS [32].

D. The Gaia Methodology

The Gaia methodology [9] was developed for the analysis and design of agent systems and was extended to support open multi-agent system in 2003 by Zambonelli et al. Gaia supports both levels of micro and macro development of agent systems. The micro level relates to the agent structure while the macro level relates to the agent society and organizational structure. It includes an analysis and design phase but does not explicitly support an implementation phase.

Gaia starts with the analysis phase as is given in figure 2. It aims to collect and organize the specification, which is the basis for the design of the computational organization. It then continues with the design phase, which aims to define the system's organizational structure. The definition is in terms of the system's topology and control system in order to identify the agent model and the service model. Gaia consists of two main phases: the analysis phase and design phase. The analysis phase is the set of requirements that are identified. It aims to understand the system and its structure. It includes the environmental model, preliminary role model, preliminary interaction model, and organizational rules model.

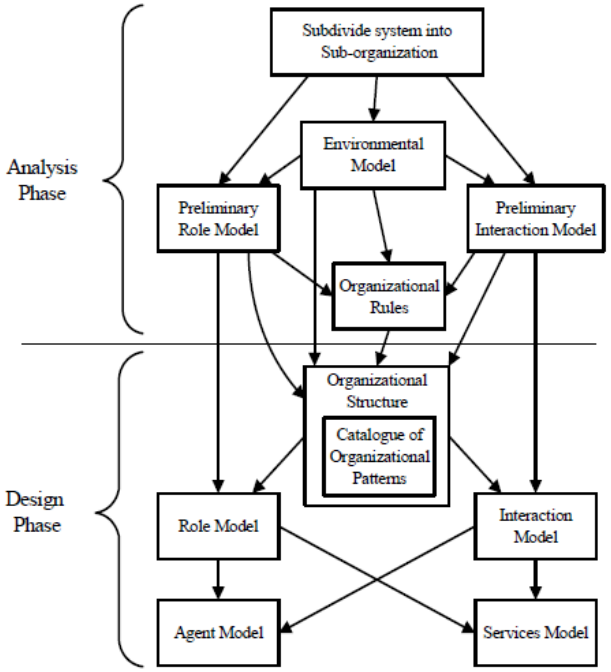

Figure 2. Gaia Methodology Models

The environmental model aims to make the characteristics of the environment explicit in which the multi-agent system will be engaged.

The preliminary role model specifies the key roles in the system and describes them in terms of permissions and responsibilities.

The preliminary interaction model captures the dependencies and relations between roles by means of protocol definitions. Gaia is only concerned with the society level; it does not capture the internal aspects of agent design.

The organizational rules model captures the basic functionalities required by the organization, as well as the basic interactions and roles.
The design phase includes organizational structure, agent model, role model, interaction model, and service model.

The organizational structure captures the catalogue's organizational patterns and involves considering: (i) the organizational efficiency, (ii) the real-world organization (if any) in which the MAS is situated, and (iii) the need to enforce the organizational rules.

The role and interaction models are the completion of the preliminary role and interaction model. This is based upon the adopted organizational structure and

involves separating, whenever possible, the organizational-independent aspects (detected from the analysis phase) from the organizational-dependent ones (derived from the adoption of a specific organizational structure). This separation promotes a design-for-change perspective by separating the structure of the system (derived from a contingent choice) from its goals (derived from a general characterization).

The agent model is concerned with identifying the agent classes that will make up the system and the agent types that will be instantiated from these classes.
The service model is concerned with identifying the services associated with a role. It identifies the main services intended as coherent blocks of activity in which agents will engage. These services are required to realize the agent's roles and their properties.

E. The Passi Methodology

It was developed in 2002 by Cossentino and Potts, is an object oriented-based methodology. PASSI is composed of five models that address different design concerns and twelve steps in the process of building a model. It uses UML as the modeling language because it is widely accepted both in the academic and industrial worlds. Its extension mechanisms facilitate the customized representation of agent-oriented designs without requiring a completely new language. Extension mechanisms here refer to constraints, tagged values, and stereotypes. The models and phases of PASSI are (see figure 3):

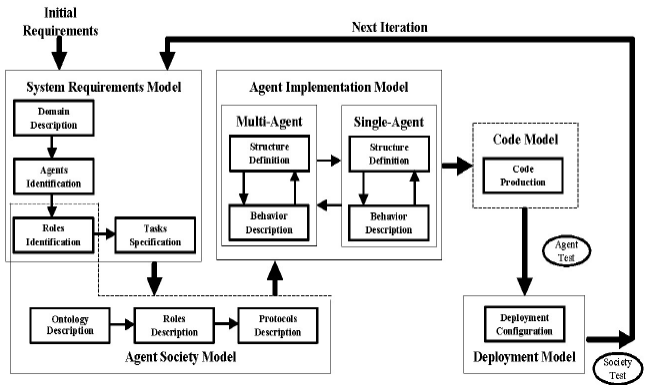

Figure 3. Models and phases of the PASSI methodology

System requirements model is an anthropomorphic model of the system requirements in terms of agency and purpose. Developing this model involves four steps:

1. Domain Description: is a functional description of the system composed of a hierarchical series of use case diagrams. Scenarios of the detailed use case diagrams are then explained using sequence diagrams.
2. Agent Identification: The separation of responsibility into agents, represented as stereotypical UML packages. In this step, one or more use cases are grouped into stereotyped packages to form agent identification diagram.
3. Role Identification: The use of sequence diagrams to explore each agent's responsibilities through role-specific scenarios.
4. Task Specification: Specification through activity diagrams of the capabilities of each agent.

Agent society model is a model of the social interactions and dependencies among the agents involved in the solution. Developing this model involves three steps in addition to a part of the previous model:

1. Role Identification: See the System Requirements Model.
2. Ontology Description: The use of class diagrams and Object Constraint Language (OCL) constraints to describe the knowledge ascribed to individual agents and the pragmatics of their interactions.
3. Role Description: The use of class diagrams to show distinct roles played by agents, the tasks involved that the roles involve, communication capabilities and inter-agent dependencies.
4. Protocol Description: The use of sequence diagrams to specify the grammar of each pragmatic communication protocol in terms of speech-act performatives like in the AUML approach [2].

Agent implementation model is a model of the solution architecture in terms of classes and methods, the development of which involves the following steps:

1. Agent Structure Definition: The use of conventional class diagrams to describe the structure of solution agent classes.
2. Agent Behaviour Description: The use of activity diagrams or state charts to describe the behaviour of individual agents.

Code model is a model of the solution at the code level requiring the following steps to produce:

1. Code Reuse Library: A library of class and activity diagrams with an associated reusable code.
2. Code Completion Baseline: The source code of the target system.

Deployment model is a model of the distribution of the parts of the system across hardware processing units

and their migration between processing units. It involves one-step:

1. Deployment Configuration: The use of deployment diagrams to describe the allocation of agents to the available processing units and any constraints on migration and mobility.

Testing: the testing process has been subdivided into two different steps:

1. The (single) agent test is devoted to verifying its behaviour concerning the original requirements of the system solved by the specific agent.
2. The society test is used for the validation of the correct interaction of the agents, in order to verify that they concur in solving problems that need cooperation.

F. The MAS-CommonKADS Methodology

MAS-CommonKADS [30] is one of the methodologies that are based on the knowledge engineering-based approach. It is considered an extension of the CommonKADS methodology [32].It consists of three main phases: conceptualization, analysis, and design.

These phases comprise of seven models that cover the main aspects of the development of multi-agent systems. Figure 4 illustrates the models of the MAS-Common KADS methodology.

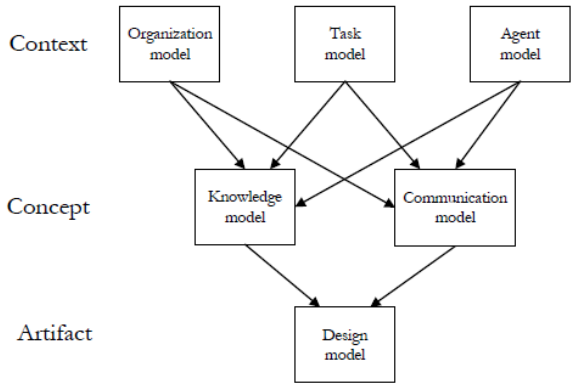

Figure 4. Models and phases of the MAS-common KADS Methodology

The methodology starts with a conceptualization phase, which is an informal phase for collecting the user requirements and obtaining a first description of the system from the user's point of view. The use cases technique is used and the interactions of these use cases are then formalized with MSC (Message Sequence Charts). The analysis and design phases define models as described below. For each model, the methodology defines the system components (constituents "entities to be modeled") and the relationships between these components. The methodology defines a textual template for describing every constituent and a set of activities for building every model. This is based on the development state of every constituent (empty, identified, described, or validated). These activities facilitate the management of the project.

The following models represent the extension of CommonKADS:

Agent model: The agent model specifies the characteristics of an agent including reasoning capabilities, skills (sensors/effectors), services, goals, etc. The agent model plays the role of a reference point for the other models. An agent is defined as any entity (human or software) capable of carrying out an activity. The identification of agents is based on the use cases diagrams generated in the conceptualization. Such identification could be augmented in the task model.

Task model: Describes the tasks (goals) that the agents can carry out. UML Activity diagrams are used to represent the activity flows and the textual template to describe the task (name, short description, input and output ingredients, task structure, etc.).

Expertise model: Describes the knowledge needed by the agents to carry out the tasks. The knowledge structure follows the KADS approach. It distinguishes domain, task, inference, and problem-solving knowledge. Several instances of this model are developed for modeling the inferences on the domain, on the agent itself and on the rest of the agents.

Coordination model: Describes the conversations between agents. That is agents' interactions, protocols, and required capabilities. The coordination model provides two milestones. The first milestone is concerned with identifying the conversations and the interactions. The second milestone is concerned with improving these conversations with more flexible protocols such as negotiation, identification of groups, and coalitions.

The interactions are modeled using the formal description techniques MSC and SDL (Specification and Description Language).

Organization model: Describes the organization in which the MASs are going to be introduced and the organization of the agent society. It illustrates the static or structural relationships between the agents. This

model also describes the agent hierarchy, the relationship between the agents and their environment, and the agent society structure. A graphical notation based on OMT is used to express these relationships, adding a special symbol in order to distinguish between agents and objects.

Communication model: Several agents can be involved in a task. This model helps with modeling the communicative transactions between systems involved. These are often human-to-system and system-to-human communications.

Design model: The design model includes the design of relevant aspects of the agent network, selecting the most suitable agent architecture and the agent development platform. The design model assembles the agent, task, expertise, coordination, organization and the communication models. This assembled collection, is subdivided by the design model to generate three sub-models:

- Application design: composition or decomposition of the agents of the analysis, according to pragmatic criteria and selection of the most suitable agent architecture for each agent.
- Architecture design: designing of the relevant aspects of the agent network: required network and knowledge.
- Platform design: selects the agent development platform for each agent architecture.

III. THE EFFECTS AND INFLUENCE OF THE APPLICATION'S DOMAIN

There does seem to be a consensus among scholars and researchers that the area of application should be considered when selecting a methodology. This is particularly apparent when the domain is well understood and defined upfront when the selection processes occur.

Our literature depicts that, there exist some aspects of agents which have been addressed in KE methodologies that are not addressed in object-oriented methodologies; For instance techniques for modeling the mental states of agents. Moreover, the social relationship between agents can hardly be captured using object oriented methodologies.

In some instances, the domain of the application has a clear and obvious influence on the type of methodology selection. In other cases, the impact seems minimal if any exists at all. This factor will serve on a case-by-case basis.

IV. RELATIONSHIP OF THE METHODOLOGIES AND THE APPLICATION DOMAIN

Relatively simpler agent based system are supported by one or a simple methodology whereas relatively complex Agent-based systems may be supported hybrid or multiple methodologies. We present the view that the more complex an Agent-based system is, the more sophisticated the methodology to design such systems must be. At present, there are no consensus standards on what methodologies are ideal for what application domains; and that majority of the studies on application domains and Agent-based solutions were used provide a way to gain insights on what attributes are useful in leading to better design methodologies.

V. ADVANTAGES AND DISADVANTAGES

The best solution to profit from the abilities of agent orientation is to study the overall of issues on subject, consider each strengths and weaknesses, and derive an informed choice. In terms of selecting a suitable AOSE methodology, the advantages and disadvantages of each methodology should be measured against the goals and application.

A. Gaia

The inherent strengths of Gaia include but are not limited to the following: (1) it does not refer to the implementation issues, thus is not limited to the specific language of a platform. (2) It could be mapped to the life cycle introduced by ESA (European space agency). (3) Both phases of Gaia (analysis and design) have deliverables (models). Gaia can be used in creating new software, re-engineering and designing systems with reuse components [9].

B. Passi

The inherent strengths of Passi include but are not limited to the following: (1) it is supported by a CASE tool, (2) Passi supports all multi-agent concepts except for the mental notion (i.,e BDI), With respect to modeling and notation aspect, it uses UML; supports well accessibility, expressiveness and complexity management [15].

C. MAS-CommonKADS

The main strength of this methodology is its simplicity, which is, extending in a natural way the standard software engineering methods. It takes into account reusability at all levels of the models, making it easy to reuse analyses and design from previous projects.

Disadvantages

D. Gaia

This methodology suffers from a number of limitations that may undermine the possibility of its effective adoption for a majority of real-world multi-agent scenarios.

The Limitations include but not limited to the following; (1) it covers the design phase but does not explicitly support an implementation phase. (2) Gaia is only concerned with the society level; it does not capture the internal aspects of agent design. (3) Gaia, in the original proposal, is suitable only for the analysis and design of closed MAS, in which agents must be benevolent to each other and willing to cooperate. Four, the notation used by Gaia to model and represent a MAS and its component appear unsuitable to tackle complexities of the real world systems and even worse, do not follow accepted software engineering methods. Five, Gaia does not address implementation and there is no tool support that we are aware of.

E. Passi

The limitations inherent in PASSI include; Multiplicity problem (from UML): the need to concurrently refer to different models in order to understand a system and the way it operates and changes over time is a critical issue. (From UML) Each model introduces its own set of symbols and concepts, thus leading to an unnatural complexity in terms of vocabulary. It does not consider the environment. It is not suitable in managing complexity.

F. MAS-CommonKADS

The limitations inherent in MAS-CommonKADS include; it offers limited support in design, testing, and coding. CommonKADS was not designed for developing MAS. The main restrictions for the direct application of CommonKADS to MAS come from the CommonKADS CM. The CM deals mostly with human-computer interaction and is very restrictive for computer-computer interaction.

VI. CONCLUSION

It is a quite difficult venture to select a specific methodology in order to employ it or even to evaluate them. This is because they usually differ in their premises, covered phases, models, concepts and the supported multi-agent system properties.

A discussion about their advantages and difficulties shows a lot of divergences; while some methodologies allow for the idea of a society of agents or the idea of an organization that offers a coherent conceptual infrastructure for analysis and design of multi-agent systems, others don't. While some methodologies explicitly provide the cooperation between agents and the concepts used to describe the type of control, others are not clear. While some methodologies are becoming close to a complete methodology for multi-agent systems, others are not. Some deal with inter-agent perspectives, others with intra-agent perspectives, and a few others deal with both.

Our literature suggests that understanding the approach a methodology takes is fundamental and especially in cases where a hybrid may be ideal.

Having considered all surrounding issues, the choice of a specific methodology by a practitioner, depends on three issues; which are (1) the approach a methodology takes e.g. Object oriented, (2) maturity, availability of documentation and tool support, and (3) possibility of co-existence of a hybrid.

VII. REFERENCES

[1] P. Bresciani, A. Perini, P.Giorgini, F.Giunchiglia, J. Mylopoulos, Tropos: An Agent-Oriented Software Development Methodology (Autonomous Agents and Multi-Agent Systems), Volume 8, Number 3, (2004) 203—236.

[2] B. Bauber, J. Odell. UML 2.0 and agents: how to builds agent based system with new UML standard, Journal of engineering applications of AI 18(2) (2005).

[3] M. Luck, P. McBurney, C. Preist, Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based

Computing), AgentLink, 2003, ISBN 0854 327886.

[4] B. Henderson-Sellers, P. Giorgini (Eds.), Agent-Oriented Methodologies, Idea Group Publishing, Hershey (2005) 107-135.

[5] F. Bergenti, M.-P. Gleizes, F. Zambonelli (Eds.), Methodologies and Software Engineering for Agent Systems, Kluwer Academic Publishing (New York), 2004.

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

[7] S. A. DeLoach, J. C. Garcia-Ojeda, O-MaSE: A customizable approach to developing multiagent development processes, International Journal of Agent-Oriented Software Engineering 4 (2010) 244–280. doi:10.1504/IJAOSE.2010.036984.

[8] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, Tropos: An agent-oriented software development methodology, Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 8 (2004) 203–236.

[9] F. Zambonelli, N. R. Jennings, M. Wooldridge, Developing multiagent systems: The Gaia methodology, ACM Transactions on Software Engineering and Methodology 12 (3) (2003) 317–370. doi:10.1145/958961.958963.

[10] L. Padgham, M.Winikoff, Developing intelligent agent systems: A practical guide, John Wiley & Sons, Chichester, 2004, ISBN 0-470-86120-7.

[11] J. Pavon, J. J. Gomez-Sanz, R. Fuentes, The INGENIAS methodology and tools, in: B. Henderson-Sellers, P. Giorgini (Eds.), Agent-Oriented Methodologies, Idea Group Publishing, (2005) 236–276.

[12] J. Pav´on, J. G´omez-Sanz, Agent oriented software engineering with INGENIAS, in: V. Mar´ık, M. Pechoucek, J. M¨uller (Eds.), Multi-Agent Systems and Applications III, Vol. 2691 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, (2003) 394–403.

[13] R. Cervenka, I. Trencansky, AML The Agent Modeling Language: A Comprehensive Approach to Modeling Multi-Agent Systems, Birkh¨auser, 2007, ISBN 978-3-7643-8395-4.

[14] M. Cossentino, V. Seidita, PASSI2 - going towards maturity of the PASSI process, Technical Report RT-ICAR-PA-09-02 (December 2009).

[15] M. Cossentino, From requirements to code with the PASSI methodology, in: B. Henderson-Sellers, P. Giorgini (Eds.), Agent-Oriented Methodologies, Idea Group Inc., (2005) 79–106.

[16] M. Elammari , W. Lalonde, :An agent –Oriented Methodology: High- Level and Intermediate Models HLIM. In Proceedings of AOIS, Heidelberg (1999)

[17] S. Munroe, T. Miller, R. A. Belecheanu, M. P˘echou˘cek, P. McBurney, M. Luck, Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents, Knowledge Engineering Review 21 (4) (2006) 345–392. doi:10.1017/S0269888906001020.

[18] L. Padgham, M. Winikoff, Prometheus: A methodology for developing intelligent agents Agent-oriented software engineering III, (2002)174-185.

[19] A.Omicini, Societies and Infrastructures in the analysis and Design of Agent-Based Systems. P. Ciancarini, and M. Wooldridge (Eds.) Agent Oriented Software engineering, Springer-Verlag, (2001) 185-193.

[20] G.Bush, S.Cranefield, M. Purvis: The Styx agent methodology (Information Science Discussion Papers Series No. 2001/02). University of Otago. Retrieved from http://hdl.handle.net/10523/831, (2001).

[21] A .Collinot, P .Carle, K .Zengal : A Method for designing computational organizations, In Proceedings of the First International Workshop on Decentralized Intelligent Multi-Agent Systems, Poland, (1995).

[22] M. Gervais, ODAC: An Agent-Oriented Methodology based on ODP. Journal of Autonomous Agents and Multi-Agent Systems, 7(3), (2003) 99-228.

[23] S. DeLoach: The MaSE Methodology. In F. Bergenti, M.P. Gleizes & F.Zambonelli (Eds.), Methodologies and Software Engineering for Agent Systems: The Agent Oriented Software Engineering Handbook, Kluwer academic publishing, (2004) 107-125.

[24] J. Lind: Iterative software engineering for multiagent systems, The MASSIVE Method, Berlin: Springer-Verlag, (2001).

[25] F. Brazier, B. Dunin-Keplicz, N. Jennings, J. Treur ,DESIRE: modelling multi-agent systems

in a compositional formal framework, Int J. Cooperative Inf. Syst. 6(1), (1997) 67-94.

[26] D. Kinny, M. Georgeff, A. Rao ,A methodology and modeling technique for systems of BDI agents, In Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a MultiAgent World (MAAMAW-96), Netherlands: Springer, (1996) 56- 71.

[27] E. Kendall, M. Malkoun , C. Jiang, A Methodology for Developing Agent Based Systems for Enterprize Integration, in P. Bernus and L. Nemes, editors, Modelling and Methodologies for Enterprise Integration, Chapman and Hall, 1996.

[28] B. Burmeister, Models and Methodology for Agent-Oriented Analysis and Design, in Fischer K. editor, Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Artificial Intelligence, DFKI document D-96-06, http://www.dfki.uni-kl.de/dfkidok/publications/D/96/06/abstract.html, 1996.

[29] B. Moulin and L. Cloutier, Collaborative work based on multiagent architectures: A methodological perspective. In Fred Aminzadeh and Mohammad Jamshidi, editors, Soft Computing: Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence, Prentice-Hall, (1994) 261–296.

[30] C. Iglesias, M. Garijo, J. Gonzalez,J. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In AAAI'97 Workshop on Agent Theories, Architectures and Languages, Providence, RI, July 1997. ATAL. (An extended version of this paper has been published in INTELLIGENT AGENTS IV: Agent Theories, Architectures, and Languages, Springer-Verlag, (1998).

[31] C. Iglesias, M. Garijo, J. Gonzalez .A Survey of Agent-Oriented Methodologies, in Intelligent Agents IV: Agent Theories, Architectures, and Languages, 1555 of LNAI, Springer-Verlag (1999) 317-330.

[32] N. Glaser .The CoMoMAS Methodology and Environment for Multi-Agent System Development, Proceedings of the Second Australian Workshop on Distributed Artificial Intelligence: Multi-Agent Systems: Methodologies and Applications, Springer-Verlag (1997) 1-16.

[33] M. Wooldridge and R. Jennings, Pitfalls of Agent-Oriented Development, in Proceedings of the Second International Conference on Autonomous Agents (Agent 98), Minneapolis/St. Paul: ACM Press, (1998) 385-391.

[34] J. Odell, Objects and Agents Compared, in Journal of Object Technology, 1 (1) (2002) 41-53.