A Landscape of Agent Systems for the Real World

Virginia Dignum

Frank Dignum

institute of information and computing sciences, utrecht university

technical report UU-CS-2006-061

www.cs.uu.nl

A Landscape of Agent Systems for the Real World

Virginia Dignum, Frank Dignum
email: {virginia, dignum}@cs.uu.nl

December 21, 2006

Abstract

Agent technology has been a promising new paradigm for the last 10 years. However, the role of agents in the implemented systems can vary considerably. It ranges from a type of expert system module to sensors in a sensor network. In this paper we will classify all these different agent systems in a few types based on the relations of the agents with their environment. We will show that each of these types of agent systems naturally requires its own type of methodologies and platforms. The classification shows some important areas for further developing methodologies and also gives some handles on how to choose the appropriate type of agent system to be designed for an application.

1 Introduction

Over the last decade, agents and multi-agent systems have moved from a niche area of artificial intelligence to become mainstream concepts in generic computing technology. However, even though the abstract notions of agency and multi-agency are subscribed by all parties, there is little consensus and research towards a formal, ontological and methodological understanding of these concepts. As Luck and d'Inverno state [14]:

> "The richness of the agent metaphor that leads to such different uses of the term is both a strength and a weakness. Its strength lies in the fact that it can be applied in very many different ways in many situations for different purposes. The weakness, however, is that the term agent is now used so frequently that there is no commonly accepted notion of what it is that constitutes an agent."

An analysis of different approaches and models shows that even if different groups use the same kind of concepts, the meaning behind those concepts is sufficiently disparate to make communication and comparison of approaches cumbersome.

A *multi-agent system* (MAS) is commonly defined as a system composed of several agents, capable of mutual interaction. The exact nature of the agents, and of their interactions is, however, subject of some controversy. Agents are commonly defined as autonomous entities, but some include characteristics such as intelligence, rationality, or mobility. The concept of interaction is also defined in different ways; so, agents can interact by message passing, or by producing changes in their common environment (stigmergy). Furthermore, notions of coordination, cooperation, collaboration, and competition are often used to extend or replace that of interaction [19]. In some approaches, MAS are seen to include human agents as well. Human organizations and society in general can in this case be considered an example of a multi-agent system. Furthermore, the organization of a MAS is also assumed in different ways. Some approaches, see organization as a feature that should be *designed* to exhibit specific characteristics, whereas others see organization as an *emergent* property of the system.

This paper aims at providing a landscape of agent system types, including a detailed description of the architecture and application of the different types. This classification will be based on a number of fundamental concepts that play a role in any agent system. These fundamental concepts will be described in section 2. Section 3 presents basic considerations on the interaction between distributed systems and their environment. In section 4 we will show how the way these fundamental concepts are modelled with respect to the agents and their environment provides a natural way of classifying agent systems into different types, each with its own type of methodologies and platforms. Section 5 will highlight some of the main characteristics of the different agent system types, while section 6 discusses the software engineering aspects related to the classification. We will highlight the different approaches and conclude that, contrarily to what is often assumed, most current approaches are not competitors of each other but aim at different sectors in the landscape, and most important, the current occupation of the landscape is not covering some important areas. There is thus room for extensions and connections between methodologies and platforms. In section 7 we indicate some heuristics that can be used to determine the most suitable agent system type for a certain situation. The paper is concluded in section 8 with some prospects for future work.

2 Principles of Agency

Many of the challenges associated with agent solutions to complex problems, refer to the distribution of tasks and the organization of the agents in a more or less known and favorable environment. Sycara has described those challenges as the decomposition of problems into individual tasks or goals for the agents; deciding on interoperability and communication options for agents; coherence of action and avoidance of conflicts and harmful or useless interaction; and individual possibilities for representation and reasoning about actions, plans and knowledge of others [21]. In the same article, Sycara also discussed the problem of engineering and constraining practical agent systems, and how to design technology platforms and development methodologies for MAS. In our opinion, the analysis of a problem domain will give rise to different decisions concerning the practical implementation of the challenges above. For example, the decision on how to conceptualize agent communication or interaction rules leads to different types of agent systems.

Independently from the type of agent system, researchers agree that a few characteristics distinguish agents from other programming paradigms. Those basic *principles* form the definition of agency, that is, describe what makes an agent an agent:

- Communication: the way the message stream is organized and resources are invoked.
- Interaction: the influence of agent actions in the environment resulting in new states.
- Goals: the organizational and individual objectives that determine agent action.
- Reasoning: the processes and strategies to evaluate goals, environment state and course of action.

Different types of agent systems are the result of architectural choices on how to model and implement these concerns. In the following, we describe these concerns in some detail and indicate how they can be used to guide the decision and determine the type of agent system.

Communication. In environments where agents are able to identify other entities as being agents as well, agents should be able to 'talk' to each other in order to decide what action to take and how that action can be coordinated with actions of others [3]. Typically, Agent Communication Languages (ACL), such as FIPA-ACL, are used for this effect. Agent communication is, in most agent systems a concern of the infrastructure in which the agents operate. This infrastructure must conform to whatever possibility is available. In the case of single agent systems, the agent in question does not assume any other 'kindred' entities in the environment and therefore has no use for an ACL. The agent design should therefore include enough communication facilities, for access to non-agent-like resources, such as procedure calls or database access. In particular, in single agent environments, the agent's only means to access other entities is through non ACL communication, that is made possible through the agent's API.

Interaction. Agents influence the environment which reacts to those influences resulting in a new state. In the same way, changes in the environment correlate to changes in the agent's internal state. How interactions are possible and how to derive the effects of interaction is determined by interaction laws, or rules. Examples of environment determined interaction rules are policies on resource access (e.g. can the agent change a database record). But agent configurations in themselves also can facilitate possibilities for interaction. In MAS interaction rules are often 'hard-wired' into the agents, especially when it concerns closed systems where all agents are known. And in open situations, participation requires that the heterogenous agents do possess the possibility of obeying the required interaction rules. In agent organizations and electronic institutions, interaction rules are fixed in the environment design which enforces compliance by the agents. Agents are, to a certain degree, free to decide on how to comply to norms and how to perform organizational tasks [4].

Goals. While all agent systems assume agents to have individual goals, Agent Organizations also assume some kind of global objectives. In fact, several authors state that agent architectures can be considered as organizations composed of autonomous and proactive agents that interact and cooperate with one another in order to achieve common or private goals [8, 5, 17]. On the other hand, [14] defines MAS as having no global system goal. That is, MAS assume goals to be a component of the agents, whereas in agent organizations goals are determined outside the agents, and their achievement dependent on agent actions. Electronic institutions' goals are also determined outside the agents, but are usually soft (e.g. fairness of exchanges) and do not require specific agent actions (e.g. no exchange at all is a fair exchange).

Reasoning. Reasoning capabilities are the main distinguishing factor of agents. It is therefore not conceivable to 'extract' reasoning from the agents themselves, while maintaining the agency paradigm. All the different agent systems types that we consider, assume reasoning (if present at all) to be part of the agents of the system.

3 Agents and Environments

Development of agent systems, as well of any other (distributed) system, is based on the assumption that there is a system which behavior and functionality can be controlled, and an external world (the environment) which is largely uncontrolled. Interfaces are responsible for the interaction between the controlled components and the uncontrollable world. When developing distributed systems, engineers are faced with two basic types of designs with respect to interaction between system components and environment. One the one hand, the different components of the distributed system (the agents in an agent system) can interact using special interaction mechanisms that make use of the knowledge of the internal functioning of the components. In agent technology we can e.g. assume all components to be (BDI) agents and use this fact to reason about the effects of communication between these agents. In order to take full advantage of this fact, specialized interaction 'channels' and languages have been developed to enable inter-agent interaction. However, this only works properly if we assume that all agents in the system are controlled by the same designer group, and thus the agents all function in a similar way.

On the other hand, application programming interfaces (API) enable components to access resources, request services, and/or exchange data. One of the primary purposes of an API is to provide standard descriptions of how (usually third party) functions can be accessed without requiring knowledge of their source code, or a detailed understanding of the functions' internal workings. Advantages are standardization of interactions and openness of systems (components can be added or deleted on the fly). Web service technology makes wide use of this type of interaction. Internet APIs provide the means for accessing search engines, web pages or images[1].

As prototypical of distributed systems, agents and environment are closely related. One of the most cited definition of agents, by Patti Maes, says: "autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed" [15].

[1]For examples see: http://code.google.com/apis.html.

The special link between agent and environment is commonly accepted as one of the features that distinguishes agents from related software paradigms, such as object-oriented systems, distributed systems, or expert systems [9, 23]. Nevertheless, the concept of environment, and more specifically, the relation between agents and their environment takes shape in many different ways. Weyns et al. provide a good overview of the different roles of environment in MAS [22]. However, in their work, they are concerned about architectures and platforms for environment design, whereas we are mostly interested in the role of the environment in the choice for a type of agent system and on the influence of the environment on agent-oriented software engineering (AOSE) methodologies.

The current practice in MAS typically associates environment with resources that are external to the agents and their communication structures [22]. Even if this is true of traditional MAS, current developments in agent research identify a rich landscape including agent organizations, electronic institutions and agent web-services, of which the traditional MAS are only one of the fields. In the next section, we will use environment characteristics and requirements to determine the features of different fields in the landscape, and how this should guide the development of specialized design methodologies. In particular, we will consider the following approaches: single-agent systems (SAS), multi-agent systems (MAS), agent organizations (AO), electronic institutions (IE) and artificial ant-like systems (AntS). These approaches embody different perspectives and inform different software engineering methods.

The different agent system types will be further described in section 5. For this moment, we limit the description to the most significant distinguishing characteristics of the systems. SAS consider that only one entity in the whole environment is endowed with agency characteristics. MAS assume several agents in the environment, and deal explicitly with communication and interaction between specific agents in the environment. In AOs, organizations are first class entities, with agents being active entities that are able to fulfil organizational positions in order to make possible the achievement of global objectives. EIs provide highly regulated environments which fully determine all interaction between agents. The same is also the case for Ant-like systems, however in this case interaction assumes much simpler entities, whereas EI assume higher cognitive agents.

In this paper, we relate these different agent systems types to different appreciations of how the principles of agency, as described in section 2, are divided between the environment perspective and the agent perspective. Depending on domain and design requirements, the system engineer should be able to determine the most appropriate balance and as such choose the most appropriate system type. Obviously, these types and their separations are rather abstract. In real life applications, differences are less black-and-white and combinations of types are most likely.

4 Balance of Concerns

Developers of agent systems have typically a fixed number of system components that can be manipulated, while others are uncontrollable. These components are the agents, their interfaces to the external world (API), the agent communication channels, and the environment itself. The two extremes are the case in which the developer has full control over the agents' architecture and interaction, and that where agents are unknown and uncontrolled and the developer fully controls the (sub-)environment where interaction occurs. Depending on what is controlled and how those components are specified, different types of systems are possible.

In a single agent system, design and architecture of the agent is fully controlled and must deal with the four concerns described in section 2. Since SAS systems assume no other agents in the environment, the specification of communication issues is usually limited, but on the other hand, this type of systems has usually a much strong interface component, think about the HCI component of embodied agents, or the sensor and monitoring architecture of spacecraft agents such as Remote Agent. Multi-agent systems also assume full control over the agents' architectures. However, MAS often rely on the chosen platform to deal with communication (ACL) issues. System goals and interaction requirements are part of the agents' design, in the sense that each agent type 'receives' its goals and interaction protocols from the system design.

Human organizations define the roles and responsibilities for organizational participants, who are expected to bring those into action depending on the task and environmental demands. Organizational concepts have been identified as a suitable way to cope with the complexity of designing agent systems systems. However, MAS research has traditionally kept an individualistic character, and have focused on the principled construction of individual agents following an agent-centered view.

AOs and EIs take an environment-oriented view of design. By controlling the way interactions can occur in an environment, Agent Organizations are able to specify global organizational goals outside the design of the agents. Roles represent organizational positions responsible for the achievement of some part of the global goals according to the predefined interaction rules. In this way, the satisfaction of global goals can be verified, independently from the agents that will act in the system. Agents can have their own goals and use their reasoning capabilities to decide on the enactment of one or another role in the system, and use their own capabilities to determine which protocol available to them is the most appropriate to achieve the objectives of the organizational positions assigned to them. EIs go a step further on the designer's control over the environment by enabling environmental design to prescribe objectives and protocols in a way that is independent of agent goals. That is, for an EI, individual goals are not relevant and only institutional behavior is possible with the system. This makes EI very suitable to deal with open environments populated by un-trustable agents. Finally, Ant Systems can be seen as a special case of institution in the sense that environment design is fully controlled such that even very simple agents with hardly any reasoning power are able to interact to achieve the system goals.

Figure 1 depicts how the separation of concerns is taken in the 4 agent system perspectives considered. Note that while these for types are prototypical discrete points in the space, in reality distribution in this space is continuous, and most systems will fall some where in between. These different types reflect not only a different perspective on design, as discussed above, but when considering the kinds of agent applications and models currently being developed, these are also the types of systems that currently exist.

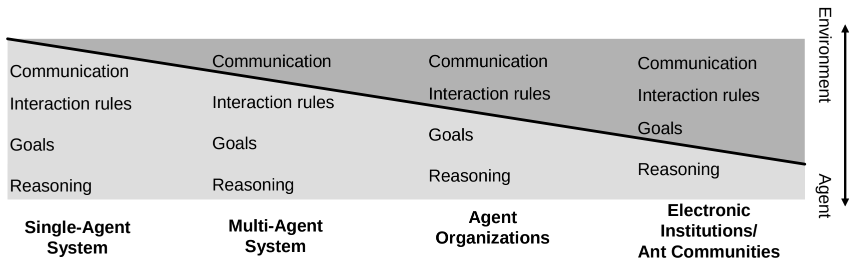

Figure 1: A sliding scale of agent system types.

Multi-agent systems form the largest group on the agent system landscape. This has an obvious 'historical' reason, but also most domains where agent applications have been deployed require a fixed number of agents with well-defined capabilities and interactions (e.g. logistics, defense, or manufacturing domains). In fact, the other types of agent systems have emerged from the need to extend the multi-agent paradigm to areas that depart from these principles, either because no specific agents can be assumed or where interactions are utterly unstable due to the dynamicity of the domain. Frameworks for design and implementation of agent systems are also mostly tailored for the development of MAS. We will discuss design methodologies further in section 6.

5 Agent Landscape

In this section, we will highlight some areas in the agent landscape has depicted in figure 1, chosen for their differentiating characteristics, based on the discussion in the previous section.

5.1 Single Agent Systems

A *single agent system* has been defined as a complex, centralized system in a domain which also allows for a multi-agent approach [20]. In such systems, all deliberation and decisions is done by one central process, the agent, even if possibly perceiving and actions can be done by several entities, connected to the agent. The agent in a single-agent system models itself (goals, actions and knowledge), the environment, and the interactions. A single agent perceives itself as the only agent in the environment, in the sense that it interacts with all components of the environment as if those are of a different *sort* than itself. If other agents exist, they are considered part of the environment, are are not modelled by the agent as having agent-like characteristics. Typical examples of single agent systems are (embodied) user interface agents, web-bots or physical robots, and agent web services.

As discussed in the previous section, autonomy is the characteristic that distinguishes agents from other (complex) systems. Agents are particularly suitable to operate in challenging environments (dynamic, unpredictable, unreliable) where failure and recovery must be performed autonomously [18]. A notorious example is Remote Agent, developed by NASA for Deep Space missions [16]. Properties of spacecraft domains are exemplar for complex domains and identify requirements for the design of agent systems:

1. Need to carry out autonomous operations for long periods;
2. need to guarantee success given tight deadlines and resource constraints;
3. high reliability, including rapid failure response under limited observability;
4. concurrent activity among tightly coupled subsystems.

Agents in a non-agent environment need to be especially adaptive, be able to anticipate and plan for contingencies, and to actively search the environment.

All communication between the agent and its environment is heterogenous, as the agent does not identify any other similar entities, with which it could communicate in an agent-like fashion. The API is therefore a very important component of single agents, as it is their only means of communication. Although it may seem that single agent systems are simpler than multi-agent systems, the contrary is often the case. In fact, often one reason to treat such a system as an agent is its intelligence and complexity. Single agent systems can thus be seen as some situated expert system.

5.2 Multi-Agent Systems

In principle, any system with more than one agent is called a *multi-agent system* (MAS). However, as recent developments have identified some special classes of multi-agent system, such as organizations and institutions, discussed in the following sections, it is necessary to present here our precise understanding of MAS. Characteristics of a MAS are: (a) each agent has bounded (incomplete) resources to solve a given problem, (b) there is no global system control, (c) data is decentralized, and (d) computation is asynchronous [13]. Social and mentalistic metaphors used to describe agents lead to design models of a high level of abstraction. From a software engineering perspective, this means that the development of MAS is not straightforward. One of the most important characteristics of MAS is that the final set of agents is not given at design time, which means that MAS are based on open architectures that allow new agents to dynamically join and leave the system, in autonomic ways not fully predictable initially [11].

Even though based on open-architectures, traditional MAS systems are often closed to truly heterogeneous agents, as the design fully specifies the agent types (or classes) and their characteristics and interface possibilities. Agent architecture implements the interaction rules, goals and reasoning mechanism of the agent, and defines as such the type of agents that exist in the MAS. Agent types and their descriptions are specified by grouping system functionalities [18]. The definition of coordination mechanisms between different agent types is a major part of the

MAS, resulting in implementations where social aspects, goals and behaviors are part of the architecture of the specific agents. Such architectures exhibit neat theoretical properties that enable the specification of elegant formalisms and design and implementation tools. The disadvantage is that systems are often closed to agents that are not able to use the same type coordination and behaviors. In practice, this means, that new agents entering the system must be instances of those classes, and thus their design is controlled by the MAS development team. Typical examples of MAS are manufacturing, logistics, defense and Cooperative Information Agents.

The agents in a MAS are aware that other entities in the environment are also agents, and the platform enables specific communication between agents. In MAS there is often limited or no interaction through API between agents, and the system as a whole often has one single API to the environment, implement as an interface agent.

5.3 Agent Organizations

Agent Organizations rely on the notion of openness and heterogeneity of MAS and poses new demands on traditional MAS models. These demands include the integration of organizational and individual perspectives and the dynamic adaptation of models to organizational and environmental changes. Organizational self-design will play a critical role in the development of larger and more complex MAS. As systems grow to include hundreds or thousands of agents, we must move from an agent-centric view of coordination and control to an organization-centric one [5]. Two main differences distinguish Agent Organizations from MAS: (1) Agent Organizations assume the existence of global goals, outside the objectives of any individual agent, and (2) Agent Organizations exist independently of agents, and do not assume any specific internal characteristics of the agents.

Agent Organizations have been advocated to deal with agent coordination and collaboration in open environments [12]. In this context organization is the set of behavioral constraints adopted by, or enforced on, a group of agents to control individual autonomy and achieve global goals. That is, one of the main issues in agent organizations is the specification of coordination mechanisms between agents playing roles in a regulated social environment. Agent organization models are typically composed of [6, 12]:

- Roles and groups: describing organizational positions
- Role dependencies: indicate some relationships between roles, through which actions can be coordinated
- Interaction structures: describe the dynamics of the organization in terms of scenes, transitions, events, etc.
- Normative structure: describe and control desired and undesired behavior
- Ontology: create common ground for communication

The system is 'open' for new agents, who often will need to register through contracts that specify their aims and capabilities. Results are determined through the combination of global and individual goals and are dependent on the capabilities of the specific agents in the population. Social behavior of the agents, e.g. how they balance own and role goals, and how much they are willing to comply to norms, are essential to determine the global behavior of the system [4]. Agent Organizations have been used to model Knowledge Management and Supply Chain domains.

5.4 Electronic Institutions

Human institutions are developed to set and enforce laws, monitor and respond to emergencies, prevent and recover from disasters, etc. Their virtual counterpart, *Electronic Institutions* (EIs), are meant to handle issues inherent to open multi-agent systems, namely heterogeneity of agents; trust and accountability; exception handling (detection, prevention and recovery from failures that may jeopardize the global operation of the system); and societal change (capability of accommodating structural changes). EIs address a restricted but significant type of openness concerning the interaction between autonomous, independent entities that are willing to conform to a common, explicit, set of interaction conventions [1].

Electronic institutions are developed as first class entities forming the environment in which agents interact. By defining the message types and common ontology that agents use to communicate, providing agent certification and registration, and fully specifying the rules of encounter, EIs are able to engender trust between heterogeneous agents. In general EIs enable to:

- Specify the co-ordination structure that is used
- Describe exchange mechanisms of the agent society
- Determine interaction and communication forms within the agent society
- Facilitate the perception of individual agents of the aims and norms of an agent society
- Enforce the organizational aims of the agent society

Typical domains for EIs are Electronic Commerce and Web based cooperative information systems. The ISLANDER framework, developed at IIIA, Spain, has emerged as the most widely known platform for EIs. It provides constructs for the core notions of IE including agents and roles, a dialogic framework, interaction scenes articulated in a performative structure, and normative rules [7].

5.5 Ant systems

Ant-like, or swarm intelligence, systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents are expected to lead to the emergence of global behavior. Ant-like systems are mostly used in the domains of social simulation, dynamic, adaptive planning, and Artificial Life. Developers typically have complete control over the design and implementation of both the environment as the agents. Agents' behavior is usually very simple, with limited or no reasoning capabilities.

Ant-like systems use mostly one mechanism for communication and co-operation: stigmergy, that is, indirect communication based on modification of the environment. If one agent reinforces a pheromone trail or deposits an object, other agents are able to perceive the modified environment, adapt their responses or start new activities. Some authors distinguish between quantitative stigmergy (the perceived concentration of a pheromone influences whether there is a reaction or not) and qualitative or discrete stigmergy (different stimuli trigger different responses). The main problem with the application of this form of co-ordination in artificial systems is its development: one agent has to modify its environment so that the others respond to stimuli in the intended way. Thus it is important to understand the corresponding mechanisms in naturally coordinated systems like social insects. Sociality is not the key concept in ant systems, but the interaction of local rules resulting in self-organized behavior or emergent patterns [2].

6 AOSE

According to the Webster's Dictionary, a methodology is defined as "a body of methods, rules, and postulates employed by a discipline", (2) "a particular procedure or set of procedures", or (3) "the analysis of the principles or procedures of inquiry in a particular field[2]. Until recently, no methodologies in this sense were available for agent-oriented development [11]. Most developers

[2]http://en.wikipedia.org/wiki/Methodology

	SAS	**MAS**	**AO**	**EI**	**Ants**
Agent Architecture	Controllable, closed environment	Controllable, closed environment	Unknown, open environment	Unknown, open environment	
Organization Architecture		Implicit: 'Sum' of the agents	Explicit, Controllable	Explicit, Controllable	
Interaction protocols		Controllable	Open to agent interpretation	Controllable	Controllable
Interaction goals		Controllable	Controllable	Controllable	Controllable
Global, external goals	No	No	Strong	Weak	
Individual, internal goals	Yes	Yes	Yes	Irrelevant	
Emergent properties desired	Possibly				Yes
System wide behavior control	No	Direct: Designed	Indirect: Norms	Indirect:Norms	Indirect: Stygmergy?
Individual behavior control	Direct: goal setting	Direct: goal setting	Indirect: role	Indirect: role	Indirect: resource allocation

Table 1: Agent system choice guidelines.

use an ad-hoc approach, which, while providing maximum flexibility, results in applications of questionable quality and provide no means for measurement and experience transfer.

Given the fact that different agent system types (as distinguished in the previous sections) differ extensively with respect to the types of concepts that are modelled in the agent and/or the environment of the agent, we can also expect that methodologies and platforms to support the different agent system types differ. Indeed if we look at the currently used platforms and methodologies we find that they do cater for different types as can be seen from figure 2. Figure 2 gives an overview of different methodologies and platforms, and their position in the agent system landscape. It also shows the existing relations between methodologies and platforms.

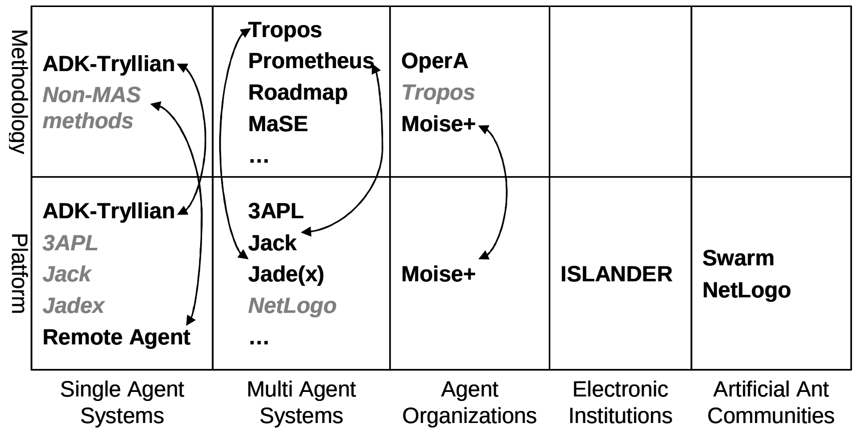

Figure 2: Overview of methodologies and platforms.

6.1 Design methodologies

In the following, we list some of the existing methodologies for each system type. This list does not attempt at completeness, but merely provides a few well-known frameworks as examples.

SAS: Interestingly, no specific methodologies are available for the design and development of single agent systems. Most agent methodologies consider systems consisting of several agents, and the communication between those agents. An exception is Tryllian's Agent Development Kit[3] that provides a platform for dynamic, smart software components that can work together with the outside world and with each other. Because in SAS systems there is (naturally) a lot of emphasis on the interaction of the agent with the environment, which in itself is quite standard, there is a tendency to use traditional OO methodologies for this part. The internal part of the agent is configured using some domain specific AI techniques. These can range from expert systems methodologies to the use of neural networks for adaptive agents. In academic settings sometimes multi-agent platforms such as 3APL or Jack, are used as well, often modified for the specific needs of the application domain. Such modifications, however, lead to low reuse and lack of transparency. The development of the Remote Agent architecture mentioned before, was based on several methodological and programming principles such as model-based programming, on-board decision and search, constraint based planning and goal-directed loop commanding [16].

MAS: Most available AOSE methodologies fall under this category. The agents are considered to be part of one, distributed system. Design starts therefore from the analysis of the overall system, while the resulting software consists mainly (if not exclusively) of the agents. The main concepts that are used in these methodologies center around goals, plans and interaction protocols. Although organizational concepts like "roles" are used, they have a different meaning in this context. They refer to specific types of functionalities that can be bundled into agent types. The overall organization of the MAS is a matter of the designer of the system and not an explicit part of the implementation. This is an obvious consequence from the fact that MAS are usually closed systems in which all participating agents are defined at design time. Because of the closed nature of MAS there is also little attention for the interaction of the MAS with its environment and most methodologies pay no attention to models of the environment in which the MAS should function. Well known examples of methodologies for MAS are Prometheus, Roadmap, Tropos, and MaSE. Some of these methodologies provide a graphical design tool for MAS models and support semi-automatic generation of agents.

AOs: Methodologies for this type of agent systems acknowledge organizations as first class citizens. In this context roles are seen as positions in an organizational structure that can be fulfilled by agents. Important is the balance between the organizational goal and the goals of the agents (or global, system goal vs. local goals). The fact that agents are seen as separate from the roles they fulfil leads to the fact that social concepts like norms are explicitly modelled. Most important examples of methodologies for this type of systems are OperA and MOISE+. Because the resulting systems are seen as "open", in the sense that agents can enter and leave the system, the methodologies for agent organizations rely on other methodologies for the design of the agents themselves. Unfortunately there are no methodologies yet that cater for designing agents that have to participate in an open MAS.

EI: In electronic institutions emphasis has shifted completely to designing interaction patterns for the agents that enter the system such that the interactions lead to certain desirable results. In some sense, this means that the goals of the agents do not influence the overall result of the system and they can be seen as being incorporated in the agent interactions. There are no complete methodologies for EIs, however there are current attempts to develop a methodology to support the design of ISLANDER applications.

Ant Systems: Methodologies for Ant-like Systems also do not exist. They should support the design of the interaction patterns from which useful organizational behavior emerges. Besides a common intuition that the interaction patterns should be based on biological mechanisms no formal theory is developed.

[3]http://www.tryllian.org

6.2 Implementation platforms.

Where there are still a number of methodologies available for all types of agent systems, there exist very little platforms to support the implementation of the agent systems.

SAS: There are some agent oriented programming languages and platforms available that can be used to program agents. Good examples are Jason, 3APL, Jade, Jadex and Jack. However, most platforms assume that the agents function as a separate application on top of an agent platform with at most a API to the rest of the world through some Java calls.

MAS: The tools mentioned with SAS are actually meant to be used for MAS. They provide standard communication infrastructure and support for the implementation of all important agent concepts such as goals, plans and protocols.

AOs: to our knowledge no 'pure' agent organization implementation platform is currently available, although an implementation of MOISE+ seems to exist

EI: Islander is the most famous example of an EI platform. The tool supports the design of dialogical scenes and controlled transition between scenes.

Ant Systems: These systems are mainly programmed in Java directly and no specific support tools for this type of systems is known to us.

6.3 Discussion

As can be seen from the figure and the descriptions above, most current work in AOSE methodologies, like presented by Henderson-Sellers and Giorgini [11], focuses exclusively on the MAS area and offers only limited support to the design of other types of agent applications. Furthermore, because the fundamental conceptual differences between agent system types highlighted in this paper are not recognized by many researchers, there is the risk that MAS approaches are taken as the only possible methodological framework, resulting in less applicable, imperfect solutions.

In the other hand, there is quite some room for developing extensions to methodologies. E.g. there is no methodology for developing agents that should participate in open agent systems. Also there appears to be a clear need to connect to traditional methodologies for the development of the environment of the agents (which can range from a communication infrastructure to a full fledged coordination facility for the agents).

Furthermore, the basic set of concepts underlying agent systems and agent-oriented methodologies is not commonly agreed upon. Moreover, even concepts which are used across communities, have in many cases different meanings, so that consensus is more difficult to achieve than one would hope. A typical case of such differences is the concept of *role*. In MAS, role is commonly associate with a set of functionalities, identifying a class of agents, while in AOs and EIs, roles refer to organizational positions. In our opinion, this makes it difficult to take a method engineering approach as suggested e.g. by Henderson-Sellers [10]. Method Engineering proposes to isolate conceptual model fragments that are coherent parts of a methodology and use those fragments as building blocks to create a methodology tailored to the specific needs of a developer and/or application domain.

7 Choosing an Agent System Type

AOSE methodologies, as presented in the previous section, support the design and specification of a given type of agent system. They do not, however, guide the choice for one or the other type of system. Faced with a specific situation, developers should be able to determine which type of agent system, i.e. what combination of agent and environment characteristics, is the most appropriate for that situation. This indicates the need for another level of methodological support, for instance in the form of a set of guidelines that support decision-making concerning the choice for a type of agent-system.

Hardly any research has been done in the area of design methodology decision support. While it is not our pretention to cover this area in the last few lines of this paper, in Table 1 we give a short overview of some issues that should be taken into account in the choice for a type of system.

In this table we highlight the characteristics of the application domain that can lead to different agent system types. Important aspects to consider are the possibility/desirability to control the design of the agents and of the overall organization; the existence of global goals external to the agents; the degree of control over the interactions; requirements related to the control of systemic and agent behavior; and, the need for emergent behavior.

Future work is required to further develop these issues, by extensive evaluation of existing systems and starting a discussion between different AOSE research groups in order to understand and make explicit the motivations and differences between approaches.

8 Conclusions

Current approaches to the design and implementation of agents and multi-agent systems exhibit large differences both at conceptual and engineering levels. In this paper, we have argued that such differences are indeed a result of different demands and characteristics of the application domain. We've classified different agent systems in a few types based on the relations of the agents with their environment. We further showed that each of these types of agent systems naturally requires its own type of methodologies and platforms. This classification shows some important areas for further development of methodologies. Although similar concepts are used by several approaches, their semantics and implementation interpretations are sufficiently different to make comparison and reuse cumbersome. That is, Method Engineering approaches are not directly applicable to generate tailored methodologies on the fly. Moreover, traditional MAS methodologies are not always directly usable to design agents for SAS, AOs or EIs. Even if there is no one-size-fits-all method, differences between methodologies should not lead to competition, but it is important for the agent community to understand that methodologies are geared to different approaches, domains and environmental contexts. There is room for more AOSE methodologies, specifically in the area of agent institutions, and for more comprehensive formalisms and metamodels to compare/classify AOSE approaches. In this paper, we've also proposed some handles on how to choose the appropriate type of agent system to be designed for an application.

Finally, it is important to realize that some important 'holes' need still to be filled in order to deploy agents in the real world. In particular, very few work has been done to develop methodologies and platforms that enable to specify agents that can join an existing AO and/or EI in a open environment, based on the agent's judgement on how to best fulfil its own goals.

Acknowledgements. This work has been partially done during a sabbatical of the authors at the University of Melbourne. The research of the first author is funded by the NWO: Veni-grant 639.021.509.

References

[1] J.L. Arcos, M. Esteva, P. Noriega, J.A. Rodrguez, and Carles Sierra. Engineering open environments with electronic institutions. *Journal on Engineering Applications of Artificial Intelligence*, 18(2):191204, 2005.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural To Artificial Systems.* Oxford University Press, 1999.

[3] B. Chaib-Draa and F. Dignum. Trends in agent communication languages. *Computational Intelligence*, 18(2):89–10, 2002.

[4] M. Dastani, V. Dignum, and F. Dignum. Role assignment in open agent societies. In *AAMAS03.* ACM Press, July 2003.

[5] V. Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic.* SIKS Dissertation Series 2004-1. Utrecht University, 2004. PhD Thesis.

[6] V. Dignum, F. Dignum, and J.J. Meyer. An agent-mediated approach to the support of knowledge sharing in organizations. *Knowledge Engineering Review*, 19(2):147–174, 2004.

[7] M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In *ATAL-2001*, LNAI 2333. Springer.

[8] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *ICMAS'98*, pages 128–135. IEEE Computer Society, 1998.

[9] S. Franklin and L. Gasser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J. Müller et al., editor, *Intelligent Agents III*, pages 21–35. Springer-Verlag, 1997.

[10] B. Henderson-Sellers. Creating a comprehensive agent-oriented methodology: Using method engineering and the OPEN metamodel. [11].

[11] B. Henderson-Sellers and P. Giorgini. *Agent-Oriented Methodologies*. Idea Group Publishing, 2005.

[12] J. Hübner, J. Sichman, and O. Boissier. S-moise+: A middleware for developing organised multi-agent systems. In O. Boissier et al., editor, *COIN I*, volume 3913 of *LNAI*, pages 64–78. Springer, 2006.

[13] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *JAAMAS*, 1(1):7–38, 1998.

[14] M. Luck and M. d'Inverno. *Understanding Agent Systems*. Springer, 2004.

[15] P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

[16] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence Journal*, 103(1–2):5–48, 1998.

[17] M. Kolp P. Giorgini and J. Mylopoulos. Socio-intentional architectures for multi-agent systems: the mobile robot control case. In *AOIS-02*, 2002.

[18] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems*. Wiley.

[19] V. Parunak, S. Bruekner, M. Fleitscher, and J. Odell. A design taxonomy of multi-agent interactions. In P. Giorgini, J. Muller, and J. Odell, editors, *AOSE III*, volume 2935 of *LNAI*. Springer, 2003.

[20] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3), July 2000.

[21] K. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

[22] D. Weyns, A. Omicini, and J. Odell. Environment as a first-class abstraction in multiagent systems. *JAAMAS*, 14(1), 2007.

[23] M. Wooldridge. Agent-based software engineering. *IEEE Proc. Software Engineering*, 1997.