Analysis and Design of Agent-Oriented Information Systems (AOIS)\(^1\)

Ofer Arazy and Carson C. Woo

Faculty of Commerce and Business Administration
University of British Columbia
ofer.arazy@commerce.ubc.ca / Carson.Woo@ubc.ca

Abstract

Analysis and design of information systems (IS) is the process of eliciting the system’s requirements and transforming them into a model that could be used to develop IS. Analysis and design of agent-oriented information systems (AOIS) relates to the very same process when it is applied to IS that are designed around the multi-agent paradigm.

A comprehensive and rigorous methodology for developing multi-agent systems is lacking [Elammari and Lalonde 1999]. Most existing multi-agent systems were developed in an ad-hoc manner. In particular, systems developers paid little attention to requirements specification and to the analysis process [Treur 1999a]. On the other hand, analysis and design methodologies have been widely developed for software engineering. They provide a formal approach to specifying the systems requirements and describing the environment where the system is to be implemented (analysis phase) and defining the architecture of the future system (design phase). There is a wide agreement on the importance of developing agent-oriented analysis and design (A&D) methodologies [Burmeister 1996, Treur 1999a, Wooldridge, Jennings and Kinny 1999, Elammari and Lalonde 1999, Luck et. al 1997, Kendall et. al 1996a], and this task is one of the top priorities in the effort of turning agent-oriented software engineering into the prevailing software engineering (SE) paradigm. Given the advancement of software engineering methodologies, especially the object-oriented approaches, it is time that we update the techniques to deal appropriately with multi-agent systems.

\(^1\) Please do not quote or cite without the authors’ permission. Working paper 99-MIS-004, Faculty of
In this paper, we describe different approaches that are suitable for analyzing and designing AOIS. Naturally, this includes agent-oriented A&D methodologies. Since A&D is basically a modelling task, ideas and models for describing agent-oriented IS are also relevant. More specifically, agent-oriented architectures, languages, and development frameworks include a representational component, and by looking into some of the representation models used in these frameworks we could gain great insight of what entities and relations should be captured by an agent-oriented A&D methodology. For this reason, we will describe the existing agent-oriented A&D methodologies as well as the representational models of several agent-oriented architectures. In addition, we analyze their strengths and weaknesses, and compare them using three different scales. Based on the analysis and the comparisons, we compile a list of requirements for and sketch an ideal unified agent-oriented modelling methodology.

The result of this work provides a map that can be used by both system developers and researchers. System developers can use the map to choose a methodology that best suits their needs; computer science and MIS researchers can use it as an introduction to the field or as an aid in developing new agent-oriented analysis and design methodologies.

1. Background

Agent-oriented approaches represent an emerging paradigm in software engineering. This new paradigm, which is built around a new type of abstraction – an agent, seems to reshape the way new information systems are designed and developed. According to this new paradigm, information systems (IS) are designed as a collection of software agents, which interact with each other in order to realize their goal, thus form a multi-agent society.

In order to apply the agent paradigm in large and complex industrial applications, and across various domains, tools for creating such systems are needed. One of the basic and most important tools in software development is an analysis and design (A&D) methodology. An A&D methodology supports the developer in the early stages of the software engineering (SE) process - the analysis and design phases.

Commerceand Business Administration, University of British Columbia, Vancouver, B.C., Canada.
There are numerous definitions of agents and multi-agent systems\(^2\). For the purpose of this work, which is to review agent-oriented software engineering methodologies, we will consider the various notions of agents by describing their most important characteristics. Most agent definitions comply with Wooldridge and Jenning’s definition of weak agency [Kendall et. al 1996a, Wooldridge and Jennings 1995b], which describe an agent with the following characteristics:

a) *autonomous* – agents operate without direct intervention;

b) *social* – agent interact with other agents thus form a multi-agent society;

c) *reactive* - agents perceive the environment, respond to changes in the environment and affect the environment through their actions;

d) *pro-active* – the agent’s behavior is goal-oriented.

Other characteristics of agents may appear in only some of the agent-oriented frameworks, and amongst them are rationality, having mentalistic notions (e.g., desires, intentions, goals, and knowledge), veracity, mobility, adaptability, and learning capability. Gasser [Briot and Gasser 1998] claims that what differentiates agents from other concepts (such as distributed objects) is the agent’s “structured persistent action” – it’s ability to strive for a goal persistently, while making choices at the time of action.

To further draw boundaries and limit the scope of this work, we will relate only to operational information systems that are used to support businesses and to the analysis and design of those systems. However, the A&D methodologies described in the following sections could easily be adapted to other applications domains such as chemistry, biology, neural sciences, or astronomy. Throughout this paper we use the terms 'organization' and 'business processes' to describe the environment where the information systems will be implemented.

When reviewing existing agent-oriented software engineering literature, it is often unclear what phase within the software engineering life cycle a particular methodology is intended to support,

what is the main aim of the framework (e.g., to provide guidance in developing applications, to solve problems, or to study social phenomena), and in what application domains the methodology should be used. To overcome these limitations, in this paper, we classify methodologies according to several dimensions, and thus provide a map for people working or intended to work in this field. There is no best A&D technique and each technique is focused on a different aspect of the system. The map presented in this paper should assist system analysts and designers to choose a methodology that best suits their needs, in addition to understanding the shortcoming and advantages of different approaches.

The scope of this work will cover techniques and methodologies that support the development (mainly in its early phases - A&D phases) of multi-agent systems for organizational applications.

This paper is different from the surveys done by Eric Yu [Yu 1997] and by Iglesias, Garijo and Gonzalez [Iglesias et. al 1999]. Eric Yu described only methodologies that support the analysis (requirements engineering) phase, and not the design or implementation phases. Iglesias et. al did not focus on analysis and design methodologies. Their survey included mainly implementation architectures and frameworks that are aimed at assisting the developers at the implementation phase. In addition, the dimensions that are used in this paper are different and larger in number, and thus create a more detailed map of the field. Last, the rapid changes in this field and the introductions of new methodologies call for an updated survey.

The paper proceeds as follows: chapter two describes the requirements of an agent oriented analysis and design methodology, chapter three lists the various analysis dimensions, chapter four analyses the A&D methodologies according to these dimensions, and chapter five sketches some guidelines to an ideal Unified Agent-oriented Modelling Language – UAML. Appendix A includes a description of each one of the methodologies analyzed earlier.

---

3 Only agent-oriented methodologies are included in the survey. Methodologies such as F3 [Bubenco 1993], which are not strictly agent-oriented, are not included, although they could be used to model some aspects of multi-agent systems. Surveys of Object-Oriented analysis and design methodologies, on the other hand, can be found in [OMG 1992, Fowler 1992, Brinkkemper et. al 1995, Arnold P. et. al 1991, Monarchi et. al 1992]

4 The term UAML is borrowed from [Treur 1999a]
2. Agent-Oriented Analysis and Design Methodologies

An analysis and design methodology supports the processes of eliciting the system requirements, analyzing the organizational environment, and designing the information system. The early requirements and analysis phases deal with specifying the requirements of the system, while the design phase deals with the detailed specification of how the system will accomplish the requirements. In the analysis phase the application domain is modeled using abstract notions, and the conceptual model is generated. The conceptual model is a model of the organization. In the design phase the information system itself is modeled using concrete concepts, which relate directly to components of the software system. The design model should provide detailed information of how the system would look like, but it should not provide instruction of how to implement the design. The design model is similar to an architect’s plan – it describes the exact form of the end product, without specifying the technique and methods that should be used to realize this plan [Mylopoulos 1999, Wooldridge, Jennings and Kinny 1999, Wand and Weber 1990, Wand and Weber 1993]. Although various analysis and design methodologies use various names to describe the models and the phases, there is generally an agreement on the structure of the A&D process. It is seen as a collection of transformations, from the very abstract model (the conceptual model) to the most concrete model (the detailed design model), where each transformation shrinks the space of possible end products, and introduces more and more implementation bias. Ideally, the transformations should be straightforward, i.e., leave no degrees of freedom to the designer, but this is not the case in most methodologies [Wooldridge, Jennings and Kinny 1999, Wand and Weber 1990, Wand and Weber 1993].

The importance of a structured A&D process, which can be found in most systems analysis and design books (e.g., [Kendall and Kendall 1999]), is twofold:

1. During development: a formal analysis and design process ensures that the system is developed according to its requirements. It makes implementation easier, and allows for fewer mistakes. The A&D process turns software engineering into a science and not an art.

---

5 The term ‘conceptual model’ has been used in the literature in different contexts. We adopt the definition of Wand and Weber [Wand and Weber 1989, Wand and Weber 1993, Wand and Weber 1995], which relates conceptual model as a model of the real-world environment (and not a model of the information system).
A&D methodologies provide the system analysts, designers, and programmers a common language to exchange ideas about the system and share common work processes.

2. During maintenance: the analysis and design models capture the early requirements that shaped the system, as well as the ideas and constraints that directed the system designers. When there is a need to change or re-evaluate the information system, or migrate it to a different platform, the documented A&D models allows programmers to locate the exact spot where the modification is needed, without having to digest thousands of code lines.

Although A&D methodologies have been used in software systems engineering for a long time and current object-oriented techniques have matured and gained wide acceptance, they are unsuitable for developing agent-oriented information systems. The reason is that agent-oriented systems use different abstractions, and the definition of an object cannot capture the structure and behavior of agents. The distinctions between objects and agents have been pointed out by many. According to [Wooldridge, Jennings and Kinny 1999]), the important ones are the agent’s flexible, autonomous problem solving behavior (the ability of the agent to reason about its environment and choose an action that draws him closer to achieving his goal) and the richness of agents’ interactions. An intuitive way of describing the differences is the slogan “objects do it because they have to, while agents do it because they want to (or choose to)”.

Due to the aforementioned differences between objects and agents, object-oriented methodologies cannot be used as is to develop agent-oriented information systems. While some ideas from object-oriented A&D techniques can be borrowed, there is clearly a need for new methodologies, which are rich enough to capture the structure and behavior of the individual agent (e.g., cognition, intelligence, rational, and emotions) and those of the agent society (e.g., trust, fraud, commitment, and social norms).

---

3. The Analysis Dimensions

This section presents the dimensions by which the various methodologies will be compared and graphically positioned. Each dimension represents a different perspective, and hence adds to the understanding of the domain. However, the dimensions are not orthogonal.

The methodologies, which will be analyzed in this paper, are listed in the table below (in alphabetical order). The table includes A&D methodologies, as well as agent-oriented architectures and frameworks. This is because there are overlaps between the development architectures and the A&D methodologies. A&D is basically a modelling task, and since agent-oriented architectures, languages, and development frameworks include a representational component, we could gain great insight of what entities and relations should be captured by an agent-oriented A&D methodology by looking into some of the representation models used in these frameworks. The classification, according to the dimensions described in the following sections, distinguishes between the aims and capabilities of the different techniques:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Methodology</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bradshaw et al</td>
<td>KaOS</td>
<td>Bradshaw et. al 1997</td>
</tr>
<tr>
<td>Brazier, Dunin-Keplicz, Treur, van Eck and others</td>
<td>DESIRE</td>
<td>Brazier et. al 1997b, Research Group AI, Vrije University, Brazier et. al 1997a, Jonker and Treur 1998</td>
</tr>
<tr>
<td>Brumeister</td>
<td>Agent-Oriented Analysis and Design (AOAD)</td>
<td>Burmeister 1996</td>
</tr>
<tr>
<td>Coltell and Chalmeta</td>
<td>CafeAGS</td>
<td>Coltell and Chalmeta 1999</td>
</tr>
<tr>
<td>Collinot, Drogoul and Benhamou</td>
<td>Cassiopeia</td>
<td>Collinot et. al 1996</td>
</tr>
<tr>
<td>Decker &amp; Lesser</td>
<td>TAEMS</td>
<td>Decker 1995, Prasaad at. al 1996</td>
</tr>
<tr>
<td>DeLoach</td>
<td>MaSE</td>
<td>DeLoach 1999</td>
</tr>
<tr>
<td>Elammari &amp; Lalonde</td>
<td>High Level / Intermediate Models</td>
<td>Elammari and Lalonde 1999</td>
</tr>
<tr>
<td>Ferber and Gutknecht</td>
<td>AALADIN</td>
<td>Ferber and Gutknecht 1998</td>
</tr>
<tr>
<td>Fisher</td>
<td>Concurrent METATEM</td>
<td>Fisher 1999a, Fisher 1996</td>
</tr>
<tr>
<td>Authors</td>
<td>Methodology/Modeling</td>
<td>Reference</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>Fishor et al.</td>
<td>TOVE</td>
<td>Fisher 1999b</td>
</tr>
<tr>
<td>Gadomski</td>
<td>TOGA</td>
<td>Gadomski 1993</td>
</tr>
<tr>
<td>Glaser</td>
<td>CommonMAS</td>
<td>Glaser 1996</td>
</tr>
<tr>
<td>Iglesias, Garijo, Gonzalez and Velasco</td>
<td>MAS-CommonKADS</td>
<td>Iglesias et.al 1996</td>
</tr>
<tr>
<td>Kendall, Malkoun &amp; Jiang</td>
<td>AO Methodology for Enterprise Modelling</td>
<td>Kendall et. al 1996a</td>
</tr>
<tr>
<td>Luck, Griffiths and d‘Inverno</td>
<td>ASE – Agent Simulation Environment</td>
<td>Luck et. al 1997</td>
</tr>
<tr>
<td>Moss, Gaylard, Wallis and Edmonds</td>
<td>SDML</td>
<td>Moss et. al 1998</td>
</tr>
<tr>
<td>Moulin and Brassard</td>
<td>MASB</td>
<td>Moulin and Brassard 1996</td>
</tr>
<tr>
<td>Pont and Moreale</td>
<td>Integrated Software Engineering (ISE)</td>
<td>Pont and Moreale 1996</td>
</tr>
<tr>
<td>Shoham</td>
<td>AOP</td>
<td>Shoham 1993</td>
</tr>
<tr>
<td>Sloman and Logan</td>
<td>SIM_AGENT</td>
<td>Sloman 1999, Sloman and Logan 1999</td>
</tr>
<tr>
<td>Tambe</td>
<td>STEAM</td>
<td>Tambe 1997</td>
</tr>
<tr>
<td>Verharen E. and Weigard H.</td>
<td>Language/Action Perspective</td>
<td>Verharen and Weigard 1994</td>
</tr>
<tr>
<td>Wagner G.</td>
<td>AOR (Agent-Object-Relationship)</td>
<td>Wagner 1999</td>
</tr>
<tr>
<td>Wooldridge, Jennings &amp; Kinny</td>
<td>Gaia</td>
<td>Wooldridge, Jennings and Kinny 1999, Wooldridge, Jennings and Kinny 2000</td>
</tr>
<tr>
<td>Yu, Du Bois, Dubois and Mylopoulos</td>
<td>I* and ALBERT</td>
<td>Yu et. al 1995</td>
</tr>
<tr>
<td>Yu and Schmid</td>
<td>Agent-Oriented Role-Based</td>
<td>Yu and Schmid 1999</td>
</tr>
</tbody>
</table>
As we have mentioned in the introduction, only agent-oriented techniques are included in this analysis. We tried our best to include all the available agent-oriented analysis and design methodologies as well as several of the most referenced agent-oriented implementation frameworks (architectures and tools that are used to support the implementation, but do not include models that support the analysis phase or a high-level design phase)\(^7\).

A description of the analysis dimensions is presented next.

### 3.1 The supported software engineering phase

We believe that the clearest and most important distinction among the various methodologies is the software engineering process that they support [Treur 1999a, Mylopoulos 1999]. This classification makes it clear what techniques could be regarded as A&D methodologies and what techniques are agent architectures.

Each of the techniques support one or more of the following phases:

- **Analysis** (also referred to as ‘Requirements Engineering’, ‘Strategic Modeling’ [Klusch 1999b, p.14] or ‘Enterprise Modeling’ [Fox et. al 1998]): this phase specifies the boundaries of the domain, the business goals and policies of the system, and sketches the organizational environment. The end result is a conceptual model (constructed using abstract entities and not concrete data entities) that describes the organizational processes, organizational entities, and their relations. The conceptual model should answer questions such as “How will the

---

\(^7\) Additional agent frameworks and implementation tools that do not concentrate on analysis or design issues, and are not included in this survey, are: MADE [O’Hare and Wooldridge 1992], ARCHON [Cockburn and Jennings 1996], AgentSpeak [Weerasooriya et. al 1994], JACK [Busetta et. al 1999], ADE [Kupries 1999], ZEUS [Ndumu et al 1999], KARO [Meyer 1998, ISYS Group 1999], DECAF [Graham and Decker 1999], MadKit [Gutknecht and Ferber 1999], TuCSoN [Omicini and Zambonelli 1998], Lightweight Agent Framework [Kindburg et. al 1999], SIF [Funk et. al 1998], MASSYVE [Rabelo et. al 1999]. A more comprehensive overview of agent-oriented development tools could be found in [Reticular Systems 1999] (a survey of commercial products and academic research program), and in [Wickler 1999].
new system help us realize the business goals?” “How will the new system fit in with the work processes?” “What organizational units and external entities will interact with the system?” and “How will the work processes look after implementing the system?”

- **Design:** the design phase (also referred to as ‘formal specifications’ phase) specifies the architecture of the future information system that will be built according to the definitions in the conceptual model. The end result of this phase is a design model, which is “an accurate and formal statement that expresses the desired behavior of the information system [Coltell and Chaltet 1999]. The design model should be implementation independent, and it should answer questions such as ”What data types will be used?” “What will be the information flows within the system?” and “What will be the attributes that define the internal structure of the agent?”

- **Implementation:** this phase specifies the programming languages, hardware platforms, transfer protocols, and communications protocols.

Requirements and Analysis oriented techniques represent the top-down approach, and provide the system designer tools for understanding the problem (why there is a need for the system, who will use it, etc.) before proceeding to the solution (the way in which the system is designed).

Design supporting methodologies assume that the organizational context is known, and focus on the architecture of the information system.

Implementation supporting techniques could be seen as a type of 4GL. Methodologies which support this phase are not really A&D methodologies; they are software architectures or implementation framework for fast generation of agent-oriented systems. We include them in the analysis because:

- Some of these methodologies include elements of the design model.
- They may provide some significant insights for design and implementation considerations. These insights may prove very important to the researchers who are interested in defining a new A&D methodology.
Eventually the system will be implemented, and hence understanding the implementation frameworks is important.

Although many methodologies do not make the distinction between the phases clear and use the same models to describe both abstract and concrete entities, we believe that the distinction is necessary and useful for avoiding confusions.

### 3.2 Abstraction Granularity

The granularity of, or the degree of details considered by, the methodology is the next classification dimension. Some models are more open-ended and only provide mechanism for sketching the system (either the organization using the conceptual model or the information system using the design model), while others are finer grained (e.g., include notions such as emotions, commitment, cooperation, negotiations, and conflict resolution.).

Fine grain models are restricted frameworks in that they make use of specific constructs and specific relations among them, while coarse grain models may leave a few degrees of freedom to the system designer. The advantage of fine grain models is that they are clearer and easier to implement, while open-ended models are less restrictive and could be tailored to the specific needs of the designer. A good example of these two ends is the design of the internal structure of an agent. Methodologies based on the BDI\(^8\) model [Kinny, Georgeff and Rao 1996, Georgeff and Rao 1995] define the agent’s cognitive mechanism using only three attributes: Beliefs, Desires, and Intention (BDI), while other methodologies may allow the designer flexibility in choosing the attributes and the relations between them. Approaches such as BDI, which use well-known and tractable models\(^9\), are necessary when the cognitive model of the agent must provide an optimal action plan. Less restrictive models with greater expressive power are necessary when there is a need for a more accurate cognitive model of the agent (similar to human), for instance in simulations of human behavior.

---

\(^8\) In the BDI model agents have beliefs, desires and intentions. These three constructs are used to define how agents perceive the world and the mechanism they employ to choose actions based on their knowledge of the world.

\(^9\) Tractable models are models that can be solved by computers in a finite time and using finite storage spaces.
Since a methodology may include a fine grained model of one aspect of the system (for instance the agent’s internal structure) and an open ended model of another aspect (for example the social dynamics), it is useful to further divide the domain of multi-agent modelling into several sub-categories (aspects), and then to examine the granularity of the models in each of them.

Two ways of categorising multi-agent systems are common: (a) the individual agent (the internal, micro view) vs. the system/societal of agents (the external, macro view) [Wooldridge, Jennings and Kinny 1999, Elammari and Lalonde 1999, Yu et. al 1995, Brazier et. al 1997b, Sloman and Logan 1999, Ciancarini et. al 1999], and (b) the dynamic aspects (behavior) vs. the static aspects (structure) [Research Group AI, Vrije University, Carley and Gasser 1999, Wand and Weber 1990, Wand and Weber 1995]. The integration of both categorisations forms a two by two matrix, where each of the four cells represents one aspect of the multi-agent system (a representation sub-category).

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamics</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The top left cell, *Individual Agent / Static structure*, relates to the way the cognitive (and emotional) map of the agent is structured: what constructs are used (e.g., beliefs, desires, and goals) and what are the relations between them. We will use the KaOS methodology [Bradshaw et. al 1997] to demonstrate this analysis. KaOS includes the following attributes of the agent’s cognitive map: Knowledge (Beliefs and Facts), Desires, Intentions, and Capabilities. Facts represent Beliefs in which the agent has confidence about; Facts and Beliefs may be held privately or be shared. Desires represent goals and preferences that motivate the agent to act, and Intentions represent a commitment to perform an action. However, emotions and constructs that relate to the emotional state of the agent are not included in this model.
The bottom left cell, *Individual Agent / Dynamics*, deals with the functionality of the agents, the way in which the cognitive map evolves (e.g., updates of beliefs), the reasoning mechanism the agent employs (for planning), and his learning capabilities. In KaOS, the agent’s dynamics model includes a description of how the cognitive map is updated as the agent goes through his life cycle: birth, life, and death (also a Cryogenic state). During their lives agents read, process, and send information continuously and according to the information they receive they may update their internal structure. However, there is no description of a planning or scheduling mechanism, and the agents seem to be acting directly on basis of their intentions, without considering the implications of their actions and other agents’ actions on the environment. Learning is also not part of this model.

The top right cell, *Social System / Static Structure*, relates to the structure within the agent society and covers issues such as the type of agents used (e.g., planners, middle agents, information gathering agents, and mediators), the hierarchies and structure in the society (e.g., organizational structure, role hierarchy, and goal hierarchy), the relations between constructs (e.g., task-resource, task-skills, and roles-tasks) and the normative component of the social system such as permissions (authority), rights, and norms. Using the same example, the definition of the social structure of KaOS is very loose. Five generic types of agents are defined (KaOS agent, Mediation Agents, Proxy Agent, Domain Manager, and Matchmaker) and aside from the class hierarchy, no model of how the agents relate to each other exist (such as friendship networks and authority networks). Another form of the system’s structure, which is missing in this model, is the hierarchies of tasks, skills, and resources, and the description of how all these relate to one another, to agents, and to the roles agents play.

The bottom right cell, *Social System / Dynamics*, represents the interactions and communications between agents, the way in which agents coordinate their activities, cooperate, and compete, and complex social interactions such as negotiation, teamwork, trust, and fraud. Social interactions are the main focus of the KaOS architecture, and it provides a rich and comprehensive mechanism for defining the ways in which agents interact. The interaction model includes the speech (illocutionary) acts and the sequencing of the messages. The speech acts are similar to those of KQML [Finin et. al 1997], only they are more general, and hence KaOS could be considered as a communication meta-architecture. Speech acts of any agent communication
language could be implemented within KaOS. Message sequences are organized in conversations, a pattern of messages transferred back and forth between (two) agents, which are modeled using state transition diagrams. Sequences are pre-defined, and hence when designing a KaOS architecture, the designer should elicit all the possible interaction sequences. This model assumes honest and consistent agents and does not deal with problems of fraud and trust as well as with problems of security.

Each of the methodologies we reviewed relates to at least two of these cells, and the degree in which each cell is being covered varies greatly. Some methodologies provide a comprehensive and detailed model of the cognitive structure of the agent (Individual Agent / Static structure), while others focus on the interactions among agents (Social System / Dynamics).

Note that the matrix cuts through all the software engineering phases (analysis, design, and implementation) discussed in the previous section, and each one of the phases relates to the four cells (at least to some extent).

Additional important aspects of analysis and design that could be thought of as categories, yet not included in the paper, are Time [Schbbens and Petit 1999] and Uncertainty. These two concepts relate to aspects that are present in every real-life system and every environment, and many times these concepts are represented in models of analysis and design methodologies. The form in which they are represented and the level of details depend on the needs. Methodologies that are used to support the development of applications in domain such as manufacturing require an explicit representation of time (otherwise, it is not possible to do scheduling), while applications domains such as information gathering do not. In a similar way, uncertainty is important in applications such as the simulation of human activities. Conceptual (analysis) and design models capture uncertainty by explicitly express it using a grammar.

Although these concepts are important, they are not an inherent part of multi-agent systems and the degree to which they are needed depends greatly on the application domain. Thus, the detailed analysis of the methodologies in the appendix does not include these aspects or categories.
In chapter four, we classify methodologies into different granularity classes based on an average coverage of all the cells. The classes we used are fine grain (detail models that may allow for some flexibility in defining new constructs), coarse grain (not so detail models that may allow some flexibility in defining new constructs), and open-ended (no model at all, but usually with lots of flexibility in defining constructs). We are aware that this classification is rather crude, but we believe that it provides a good starting point to look at different methodologies. Readers interested in a specific class can then refer to the appendix for a detailed analysis of those methodologies.

3.3 Conceptual foundations

Multi-agent systems are built by researchers and engineers from various fields, to achieve various objectives. Although these people use the same work processes for developing the systems, their background and the conceptual foundations they employ influence the tools and techniques they choose to use. Mapping the various A&D methodologies according to their conceptual foundations sheds a new light on the methodologies, and helps to understand why they are designed the way they are, and in what domains their use is most appropriate.

Bellow is a list of the conceptual foundations, which is not exclusive (i.e., a methodology can draw on more than one foundations):

- Software Engineering: in this category we included classical A&D methodologies (including workflow and business processes modelling techniques), which employ ideas from the system analysis and modelling fields, as well as agent-oriented programming languages (e.g., [Shoham 1993]). The A&D methodologies, for example [Wooldridge, Jennings and Kinny 1999] and [Wand and Weber 1990], emphasize ideas such as the expressive power of the model, clarity and completeness, and stress the importance of a clear and unambiguous transformation from high level models (conceptual models) to lower level models (design models). Methodologies that belong to this group rely mainly on models and concepts from object-oriented analysis, design, and programming, as they currently are the most commonly used techniques and the object-oriented framework is the closest one to agent orientation.
- Problem Solving: methodologies that rely on problem solving as a conceptual foundation are used to design multi-agent systems for optimizing some global goals. These systems are built by researchers in the field of AI, DAI, and Decision Science, to solve problems such as planning and scheduling. What distinguishes methodologies in this group is the models they build must be tractable and use formal tools. The TAEMS framework [Decker 1995, Prasaad at. al 1996] is an example of such a framework.

- Organizational and Social Sciences: theories and models from the social sciences influences greatly the design and implementation of agent-oriented information systems. Some methodologies make this link explicit and liken the process of building a multi-agent system to that of constructing a society of agents. These methodologies use models from psychology and the behavioral sciences to model the agents’ internal processes, and ideas from organizational science to design the society of agents. The frameworks used in the field of Computational Organization Theory (COT), a field of science that uses the power of computers to study and simulate human behavior, are examples of analysis techniques in this group. Although the frameworks used by COT researchers (for example the TOVE framework [Fox et. al 1998, Gruninger and Fox 1994, SDML [Moss et. al 1998]) are not generally thought of as A&D methodologies, they could easily be used to analyze any type of agent-oriented information systems. COT frameworks rely heavily on the agent paradigm and provide models to support the analysis process. However, the design process is not supported because the final outcome is simulation rather than an actual system being constructed. The methodologies that belong to this group are very expressive and include complex and comprehensive models of the agent’s cognitive behavior and the behavior of the agent society.

- Simulations and Automatic Prototype Generation: techniques and tools under this category are not interested in solving problems, nor in creating a general framework for software engineering; they are interested in fast generation of multi-agent systems. Usually these multi-agent systems are simulations that are used to simulate animal and human behaviors, or simple prototypes to test designs and architectures of agents (e.g., [Sloman and Logan 1999]).
Knowledge Engineering: Knowledge Engineering (KE) is in many ways similar to the information systems analysis and design process. Both are interested in representing real-world systems in a conceptual model (capturing both the structure and the dynamics of the system), and both transform that model into a design model, which describes the architecture of the information to be built. However, traditionally KE techniques, which are used mainly in the construction of knowledge-based and expert systems, were developed separately from analysis and design techniques. In this category we included methodologies that originated from KE.

This analysis dimension may overlap somewhat with the dimensions reviewed earlier, but it does provide some additional knowledge and understanding of the various methodologies. It is useful not only as a method for organizing this field of research, but also as a practical tool to be used by systems developers to choose a methodology that best fits their background and world-views.

4. Mapping the Methodologies

Below is a mapping of the methodologies according to two dimensions: on the x axis - the supported software engineering phase, and on the y axis - the granularity. Please note that we have classified methodologies according to the SE phase that they could be used to support, even if they were not designed as such (for example the TOVE model was developed to study and simulate social organizations, but parts of it could be used to support the analysis phase).
Is it apparent from the above analysis that only few methodologies support both the analysis and design phase, and only one methodology (i.e., DESIRE) covers the analysis, design, and implementation phases.

This graph reveals no obvious pattern and it is somewhat surprising that specifications-oriented methodologies (those that focus on the analysis phase) are not more open-ended than design-oriented and implementation-oriented frameworks.
The following table is a classification of the methodologies according to the third dimension – the conceptual foundation. Each methodology is associated with a major influence, the foundation that had the greatest effect on the methodology, and a minor one if applicable. The following abbreviations are used in the table:

- **SE**: Software Engineering Methodologies
- **PS**: Problem Solving
- **SOS**: Social and Organizational Sciences
- **SAPG**: Simulations and Automatic Prototype Generation
- **KE**: Knowledge Engineering

<table>
<thead>
<tr>
<th>Authors</th>
<th>Technique</th>
<th>Major Influence</th>
<th>Minor Influence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bradshaw et al</td>
<td>KaOS</td>
<td>SAPG</td>
<td>SE</td>
</tr>
<tr>
<td>Brazier, Dunin-Keplicz, Treur, van Eck and others</td>
<td>DESIRE</td>
<td>SAPG</td>
<td>PS</td>
</tr>
<tr>
<td>Brumeister</td>
<td>Agent-Oriented Analysis and Design (AOAD)</td>
<td>SE</td>
<td></td>
</tr>
<tr>
<td>Coltell and Chalmeta</td>
<td>CafeAGS</td>
<td>SE</td>
<td></td>
</tr>
<tr>
<td>Collinot, Drogoul and Benhamou</td>
<td>Cassiopeia</td>
<td>SE</td>
<td>SOS</td>
</tr>
<tr>
<td>Decker &amp; Lesser</td>
<td>TAEMS</td>
<td>PS</td>
<td></td>
</tr>
<tr>
<td>DeLoach</td>
<td>MaSE</td>
<td>SE</td>
<td></td>
</tr>
<tr>
<td>Elammari &amp; Lalonde</td>
<td>High Level / Intermediate Models</td>
<td>SE</td>
<td></td>
</tr>
<tr>
<td>Ferber and Gutknecht</td>
<td>AALADIN</td>
<td>SE</td>
<td>SOS</td>
</tr>
<tr>
<td>Fisher</td>
<td>Concurrent METATEM</td>
<td>SE</td>
<td>SOS</td>
</tr>
<tr>
<td>Fox et al.</td>
<td>TOVE</td>
<td>SOS</td>
<td>PS</td>
</tr>
<tr>
<td>Name</td>
<td>Method</td>
<td>SE</td>
<td>KE</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>Gadomski TOGA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glaser CommonMAS</td>
<td></td>
<td>SE</td>
<td>KE</td>
</tr>
<tr>
<td>Iglesias, Garijo, Gonzalez and Velasco MAS-CommonKADS</td>
<td>SE</td>
<td>KE</td>
<td></td>
</tr>
<tr>
<td>Kendall, Malkoun &amp; Jiang AO Methodology for Enterprise Modelling (AOMEM)</td>
<td>SE</td>
<td>PS</td>
<td></td>
</tr>
<tr>
<td>Kinny, Georgeff &amp; Rao KGR</td>
<td></td>
<td>PS</td>
<td>SE</td>
</tr>
<tr>
<td>Luck, Griffiths &amp; d'Inverno ASE – Agent Simulation Environment</td>
<td></td>
<td>SAPG</td>
<td></td>
</tr>
<tr>
<td>Minar, Burkhart, Langton &amp; Askenazi SWARM</td>
<td></td>
<td>SAPG, SOS</td>
<td></td>
</tr>
<tr>
<td>Moss, Gaylard, Wallis and Edmonds SDML</td>
<td></td>
<td>SOS</td>
<td></td>
</tr>
<tr>
<td>Moulin and Brassard MASB</td>
<td></td>
<td>SE</td>
<td>SOS</td>
</tr>
<tr>
<td>Pont and Moreale Integrated Software Engineering (ISE)</td>
<td></td>
<td>SE</td>
<td></td>
</tr>
<tr>
<td>Shoham AOP</td>
<td></td>
<td>SE</td>
<td>SOS</td>
</tr>
<tr>
<td>Sloman et al. SIM_AGENT</td>
<td></td>
<td>SAPG</td>
<td></td>
</tr>
<tr>
<td>Tambe M. STEAM</td>
<td></td>
<td>SAPG</td>
<td>SOS</td>
</tr>
<tr>
<td>Verharen and Weigard Language/Action Perspective</td>
<td></td>
<td>SE</td>
<td></td>
</tr>
<tr>
<td>Wagner G. AOR (Agent-Object-Relationship)</td>
<td></td>
<td>SE</td>
<td></td>
</tr>
<tr>
<td>Wooldridge, Jennings &amp; Kinny Gaia</td>
<td></td>
<td>SE</td>
<td>SOS</td>
</tr>
<tr>
<td>Yu, Du Bois, Dubois and Mylopoulos I* and ALBERT</td>
<td></td>
<td>SE</td>
<td>SOS</td>
</tr>
<tr>
<td>Yu and Schmid Agent-Oriented Role-Based</td>
<td></td>
<td>SE</td>
<td>SOS</td>
</tr>
</tbody>
</table>
By integrating the information from the table above with the classification of the supported SE phase (see top of section four), we are able to gain a deeper understanding of the methodologies. Methodologies that have Software Engineering as a conceptual foundation and support the analysis and/or design phases (e.g., MaSE) are strictly agent-oriented analysis and design methodologies; Methodologies that have Software Engineering as a conceptual foundation and support the implementation phase (e.g., Concurrent METATEM) are agent-oriented programming languages. Methodologies with the social and organizational sciences as a (major) conceptual foundation (e.g., SDML) are usually COT frameworks. Frameworks interested in simulations and automatic prototype generation (e.g., KaOS and SIM_Agent), as well as frameworks aimed at problem solving (e.g., BDI and TAEMS) focus mainly on design and implementation issues and not on analysis.

5. Sketching an Ideal Unified Agent-oriented Modelling Language (UAML)

There is a wide agreement on the need for a Unified Agent-oriented Modelling Language (UAML) [Treur 1999a, Luck et. al 1997, Treur 1999c], a methodology that will support fully the early requirements, analysis and design phases, and provide tools for describing the complexity of multi-agent systems. An ideal unified framework for the analysis and design of agent-oriented information systems should answer the concerns raised by Luck, Griffiths & d'Inverno [Luck et. al 1997]:

“The field of agent-oriented systems is growing dramatically in many directions. Coupled with its relative youth, however this has given rise concentration of research in distinct niches so that there are very different approaches to essentially similar problem areas with, in some cases, little or no interaction. Such a fragmentation leads to a lack of consensus regarding such fundamental notions as agents and autonomous agents, and also impedes progress toward integrated approaches to agent theory and agents construction. As the field matures, the broader acceptance of agent-oriented systems will become increasingly tied to the availability and accessibility of well-founded techniques and methodologies for system development”
The UAML should include models of both the individual agent and the agent society and of both the static structure and the dynamics of the system. When developing the future A&D methodology, several important issues should be addressed:

- **Individual Agent:** What model should be used to specify the cognitive map of the agent? What constructs should be included to describe the transformation from perceptions to actions? Should the model allow for an optimal decision-making? Should constructs that describe emotions be included?

- **Social System / Static Structure:** What models should be used to specify the system’s structure? What concepts should be included in model (for example roles, resources, task, and skills)? What are the relations between these constructs? Should the constructs and the relations that specify the system’s structure be part of the model or should the system developer decide the hierarchies (for example role and goal hierarchies) and the relations he will use? How are these hierarchies and relations relate to the individual agent and his choice of actions? What types of agents should be used (e.g., information agents, mediators, interface agents, and brokers)? Can the system developer define new types of agents?

- **Social System / Dynamics:** How should the system’s dynamics be modeled? What model should be used to describe the agents’ interactions? How should concepts such as trust, fraud, commitment, and cooperation be modeled?

- **(Optionally) Time:** What is the best way to model time? How does the notion of time reflected in each of the points mentioned above?

Obviously it is not easy to answer all these questions. However, we believe that we can provide some guidelines and ideas for interested people to develop the ultimate unified agent-oriented analysis and design methodology:

- **Start from Requirements**— represents a top-down approach, where we start with eliciting the system’s requirements (the analysis phase) and then proceed to formalising the specifications (the design phase). The methodology provided by Wooldridge, Jennings and Kinny [Wooldridge, Jennings and Kinny 1999] is an example of a requirements-driven A&D methodology. Design oriented approaches (or specifications oriented) might seem appropriate when the entire software engineering process is performed by an internal IS team.
that is familiar with the organizational environment. In these cases, the analysis phase, which is used to capture the context in which the application is developed, seems unnecessary. Implementation-driven approaches are sometime used when the applications are relatively simple and are developed by a small team, where the same people are responsible for the analysis, design, and implementation. Both these approaches (design driven and implementation driven) are clearly inappropriate for the development of large-scale industrial strength applications.

- **Employ the Sociological approach** – Bordini claims that “the cognitive approach to anthropology is argued to be a suitable theoretical foundation for this topic [the development of multi-agent systems]. Fieldwork practice in social anthropology is also indicated as a useful source of ideas [Bordini 1999]. The A&D process should be thought of as a process of constructing a society of agents [Wooldridge, Jennings and Kinny 1999, Carley and Gasser 1999, Ciancarini et. al 1999]. This approach should be employed at both the analysis phase (view organizational entities and processes as interactions between autonomous agents) and the design phase (design software entities that behave similar to human agents). Models of human cognition, human behavior, and human interactions should be used in the A&D methodology. Agents represent a new abstraction, more complex than an object, and in many ways software agents resemble human agents. Complex behavior emerge from the interactions among software agents and these behaviors could be best understood using notions such as cooperation, competition, trust, commitment, and so on – notions that are used to describe human’s social behavior. To capture and represent these behaviors, the UAML should include appropriate models. According to the sociological approach to developing agent-oriented IS, if we want to design human-like societies of agents (with the entire necessary social establishment: police, defence forces, insurance agents, etc.) we should draw on formal social theories and employ knowledge from Anthropology and Organizational sciences. The same is relevant to the design of the model of the agent’s cognitive (and emotional) structure. Approaches that are based on common sense models are fine, but ultimately we should built on complete theories of human behavior, which provide a comprehensive and detailed model of human cognition, emotion, and action to design the internal structure of an agent. Such a model should draw on knowledge from Psychology, Philosophy, Linguistics, and the Social Sciences. The STEAM framework [Tambe 1997] is
an example of a methodology that builds extensively on knowledge from psychology and formal social theories.

- **Cover and clearly distinguish all phases** – the UAML should support both the analysis and design phases, and the distinction between the phases should be clear. It should provide tools for capturing the complexity of the organizational environment (analysis/conceptual models) and for designing heterogeneous and complex multi-agent system (design models). Many methodologies do not provide guidelines on how to decide what entities should be represented by agents and what entities should not, how to decide what types of agents should be used (mediators and others) and how to proceed from the conceptual model to the design model [Wand and Woo 1999, Yu et. al 1995]. The UAML should include aside from the grammar, a clear description on how it should be used. The transformation between the conceptual models (used in the analysis phase) and the design models should be straightforward. There should be a direct mapping between concepts in the analysis model to concepts in the design model. The design model should be detailed enough to allow a programmer to implement it unambiguously. Both the analysis and the design models should provide models for describing all aspects of the system: the structure and the dynamics, the individual agent and the system as a whole. An example of a methodology that supports the analysis and design phases, enables direct transformation between the models and provides clear guidelines on how to use the models is offered by Elammari and Lalonde [Elammari and Lalonde 1999]. Another approach to providing clear and unambiguous analysis process (although not agent-oriented) is based on ontology (a branch of philosophy that deals with the definition of reality and the description of the world [Wand and Weber 1990, Wand and Weber 1993, Wand and Weber 1995]).

- **Be expressive** – the internal model of the agent should be as expressive as possible [Schbbens and Petit 1999]. Although such models might not be appropriate for allowing the agents to optimise their plans, they should still be used. Carley and Gasser [Carley and Gasser 1999] provide a map relating the agent’s cognitive architecture and his capabilities to the knowledge the agent has and the tasks he performs. This work is based on Carley and Newell’s work [Carley and Newell 1994], which argue that the cognitive architecture constrains the type of knowledge and actions the agent is capable of. This map is an excellent
source for understanding the complexity of the agent’s cognitive model and for positioning existing agent architectures.

We believe that by providing more structure and complex relations at the system level, we will create more constraints over the behavior of the agents, and thus make it easier for the agent to formulate a plan. We should not reduce the expressive power of the model just to make it tractable; the model should be as expressive as possible, and at a later stage (during implementation) techniques and heuristics for planning and scheduling in that environment should be developed. Generating an optimal plan should not be a concern at the preliminary phases, where the conceptual and design models are formulated. This promises an architecture that is independent of the agent’s reasoning model. In applications where rational agents are needed, a translation between the reasoning-independent model and the reasoning model should be developed. Again, the methodology provided by Wooldridge, Jennings, and Kinny [Wooldridge, Jennings and Kinny 1999] complies with this requirement.

We also believe that the methodology should not restrict the constructs that are used to model the agent’s cognitive map, nor should it restrict the relations between them. SIM_AGENT [Sloman and Logan 1999] is an example of such architecture.

- **Be comprehensive** – the model of the system’s structure should be comprehensive and include a variety of concepts and relations between them. “An organization design could be conceptualised as a specific configuration of parameters that control the organisation’s behavior” [Carley and Gasser 1999], and these parameters may include tasks, resources, skills, and organisational structure. The hierarchies of each parameter, and the relations between them (or concepts) restrict the system’s behavior and the behavior of each component (agent). The more parameters and relations supported by the methodology, the more able it is to capture the richness of real-life organizations. In addition, more structure introduces more constraints over the agent’s behavior, making it easier to control and optimise their behavior. The UAML should allow the developers on one hand to add parameters and relations to the model, and on the other hand to omit certain structures they may find unnecessary (without affecting the competence of the models). The TOVE [Fox et.
al 1998] framework focuses on the system’s structure and provides the most comprehensive model for describing hierarchies and relations between different constructs.

- **Use the appropriate languages** – The UAML should provide graphical diagrams to capture every aspect of the system (easy to use), textual description of each step in the process (easy to understand), and formal semantics (clear and unambiguous definitions [Schbbens and Petit 1999]).

- **Represent open systems** – describing and designing open systems poses two major challenges: (a) interactions among agents that use different languages and different ontologies, and (b) cooperation among agents that do not share a common goal.

Both of these challenges are difficult and no true and full solutions exist. To solve the first problem, translation mechanisms are used. While translating between languages is a relatively simple task, translation between ontologies poses a much greater challenge (for more details see [Genesereth 1997]). Existing solutions for the second challenge include conflict resolution mechanisms (for instance [Barber K. et. al 1999]). The UAML should be able to represent both these concepts and offer solutions [Wooldridge, Jennings, and Kinny 1999].

- **Be implementation independent** – the methodology should be independent of all aspects of implementation: hardware platforms, operating systems, transport protocols, and programming languages.

The field of Computational Organizational Theory (COT) can provide much knowledge and insight for constructing the foundations of the UAML. As mentioned earlier, models of COT could easily be applied to the requirements and analysis phases. These models comply with many of the guidelines sketched above: they are founded on psychological\(^{10}\) and social theories, requirements driven, and comprehensive. A good introduction to COT can be found in [Carley and Gasser 1999] and [Epstein and Axtell 1996].

---

\(^{10}\) The internal structure of the agent is usually based on a cognitive architecture. [University of Michigan 1999] includes a comprehensive study of the theory in this research filed, as well as description of specific architectures.
One of the organizations that have been working toward achieving the goal of developing the UAML is AgentLink, the European Network of Excellence for Agent-based Computing. AgentLink has a special interest group (SIG) on Methodologies and Software Engineering for Agent Systems. This group, headed by Jan Treur, is trying to draw a roadmap leading to the UAML. In a series of meetings [Treur 1999a, Treur 1999c] they have defined short term, medium term, and long term objectives for developing an agent oriented software-engineering methodology that will support the analysis, design, and implementation phases. However, the SIG’s work has been concentrating on defining the goals and has not yet started on designing the UAML. This paper does draw guidelines to the design of the future agent-oriented A&D methodology, and thus is complementary to the work of AgentLink and could serve as a basis for the next step in the SIG’s mission.

Another attempt to describe the approach to developing an agent-oriented software engineering methodology could be found in [Brazier et. al 1999] and in [Yu 1997].

**Concluding Remarks**

As agent technology is gaining greater acceptance and multi-agent systems are becoming increasingly prevalent, there is a growing need for tools and methodologies to support the development of agent oriented information systems.

Jan Treur states [Treur 1999a]:

“It is crucial that the basic principles and lessons of software and knowledge engineering are applied to the development and deployment of multi-agent systems. At present, the majority of existing agent applications are developed in an ad hoc fashion - following little or no rigorous design methodology and with limited specification of the requirements or design of the agents or of a multi-agent system as a whole.”

In this paper, we reviewed the existing techniques for supporting the development of agent-oriented systems. The field is currently in its early stages of development, and there is no existing methodology that can provide the full support all the way from analysis and design to implementation. The future UAML, Unified Agent Modelling Language, should integrate ideas
from existing methodologies to form a comprehensive framework for agent-oriented analysis and design.

There are several contributions of this work. First, the mapping of existing methodologies using the three dimensions allows new comers to the field, as well as veteran researchers, to understand the aim and the conceptual foundations of each methodology. The maps bring order and structure to an area where confusion of terms and definitions is widespread. Software engineers and developers can also benefit from this classification, which allows them to use the most suitable methodology for their specific task. Second the guidelines to the future UAML can serve as a starting point for researchers who are developing agent-oriented analysis and design methodologies. Finally, the collection of the descriptions of all the methodologies in the appendix can serve as a dictionary and a reference source for this field.

References


[Wand and Woo 1999] Wand and. and Woo C., Ontology-Based Rules for Object-Oriented Enterprise Modelling, 1999

[Wickler 1999] Wickler G., Tools and Software for Building Intelligent Agents,

and Practice, The Knowledge Engineering Review, 10 (2), pages 115-152, 1995

and Languages, in Proceedings of the first International Conference on Multi-Agent

Development, in Proceedings of the Second International Conference on Autonomous

[Wooldridge, Jennings and Kinny 1999] Wooldridge M., Jennings N. and Kinny D., A
Methodology for Agent-oriented Analysis and Design, in Etzioni O., Muller P. and
Bradshaw J. editors, Proceedings of the Third International Conference on Autonomous

Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents
and Multi-Agent Systems, (to appear) 2000

Pohl editors, Proceedings of the Third International Workshop on Requirements
Engineering: Foundations for Software Quality, June 16-17, 1997, Barcelona, Catalonia,

International Conference on Cooperative Information Systems (CoopIS-95), May 9-12,
1995, Vienna, Austria, pages 194-204, 1995

[Yu and Schmid 1999] Yu L. and Schmid B., A Conceptual Framework for Agent-Oriented and
Role Based Modeling, in Proceedings of the Agent-Oriented Information Systems (AOIS)
in the Third International Conference on Autonomous Agents, May 1, 1999, Seattle,
USA, 1999