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
2053 Main Mall, Vancouver, B.C., Canada V6T 1Z2
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 using the multi-agent paradigm.

A comprehensive and rigorous methodology for developing multi-agent systems is lacking [Elammar and Lalonde 1999, Odell et al. 2000]. Most existing multi-agent systems were developed in an ad-hoc manner, and systems developers paid little attention to requirements specification and the analysis process [Treur 1999a].

The paper has two goals: (a) to provide an overview and (b) to discuss challenges and future research of the field. To address the first goal, we review different methodologies that are suitable for analyzing and designing AOIS. This is done by examining, for each methodology, its suitability in supporting the early phases of software engineering process (specifically analysis and design) as well as its capabilities for modeling agent-oriented systems. To address the second goal, we analyze the limitations of existing approaches, identify critical issues, and point to what we think are possible future directions.

---

\(^1\) Please do not quote or cite without the authors’ permission. Working paper 99-MIS-004, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, B.C., Canada. This research was
1. Introduction

1.1 Background

Agent-oriented approaches represent an emerging paradigm in software engineering. This emerging 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 paradigm, information systems (IS) are designed as a collection of software agents, which interact with each other in order to realize their goal, thus forming 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.

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 to defining the architecture of the future system (design phase). Yet, there are no well-established agent-oriented methodologies. There is, however, 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, Elammarri and Lalonde 1999, Luck et al. 1997, Kendall et al. 1996]. This task is one of the top priorities in the effort of turning agent-oriented software engineering into a practical and useful approach, and agent-oriented IS into widely-adopted information systems architecture.

The purpose of this paper is, therefore, to review existing methodologies, models, and representations that are necessary or useful in guiding the development of agent-oriented information systems (AOIS).

supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.
1.2 Terminology

1.2.1 Information Systems

For the purpose of this work, we consider an information system as a computer-based system for supporting the information processing needs of the organization. It is not restricted to application systems based on database management systems (DBMS) technology. Systems such as those for supporting decision-making (e.g., expert systems and simulation systems) are within the scope, since they also support the information processing needs of the organization. Throughout this paper we use the terms 'organization' and 'business processes' to describe the environment where the information systems will be implemented, and the term ‘system’ to refer to the information system.

1.2.2 Systems Analysis and Design

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 organizational notions, and the conceptual model\(^2\) is generated. The conceptual model is a model of the organization and the business processes. In the design phase the information system itself is modeled using technical concepts, which relate directly to components of the software system. The design model should provide detailed information of what 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].

\(^2\) 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...
1.2.3 Agent-Oriented Information Systems

There are numerous definitions of agents and multi-agent systems. For the purpose of this work, 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. 1996, Wooldridge and Jennings 1995b], which describe an agent with the following characteristics:

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

b) *social* – agents interact with other agents, thus forming 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 behaviour is goal-oriented.

Other characteristics of agents may only appear in some of the multi-agent systems, and thus are not part of the weak agency definition. For example, rationality, veracity, mobility, adaptability, and learning capability.

Given the above description of information systems and agents, we consider agent-oriented information systems (AOIS) as multi-agent systems, which consist of autonomous, social, reactive, and pro-active agents, for supporting the information processing needs of the organization. For the purpose of this paper, we also use the terms ‘agent-systems’ and ‘multi-agent systems’ to refer to AOIS.

---


4 Gasser [Briot and Gasser 1998] gives a slightly different definition of agents, and 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.
1.2.4 Agent-Oriented Analysis and Design

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 using the multi-agent paradigm.

From a software engineering perspective, analysis and design (A&D) of agent-oriented information systems\(^5\) is not very different from A&D of other information systems. In both the system analyst and designer has to follow common software engineering practices and elicit requirements, represent the requirements in a conceptual model, design the system’s architecture and describe the architecture in the design model.

From a modeling perspective, A&D of agent systems is unique and is based on the agent paradigm, thus both the organizational environment and the information system are represented in terms of agents (human or software) and their interactions. The distinct characteristics of agents and agent systems (see Section 1.2.3 above) require new forms of representations.

We will revisit both these perspectives throughout this paper.

1.3 Motivation for the Paper

We believe that a survey of agent-oriented analysis and design techniques is necessary for the following reasons:

1. To date, there is no comprehensive survey that focuses on analysis and design methodologies for agent-oriented information systems. Existing surveys either studied the general analysis and design methodologies that are not strictly agent-oriented, or they studied the general agent-oriented frameworks that are not necessarily analysis and/or design

\(^5\) In this paper we also use the terms ‘agent-oriented A&D’ and ‘A&D of agent systems’
methodologies (such as agent-oriented programming languages, implementation architectures, or agent-oriented simulation environments).

2 Over the past several years, many frameworks and techniques for developing agent-systems have been proposed. This paper identifies nineteen agent-oriented methodologies and approaches for supporting analysis and/or design. The rapid growth in the field and its growing importance requires some organization in order to better understand what was achieved to date, what still needs to be done, and what are the future challenges.

3 It is unclear in existing literature whether a methodology is intended to support the analysis, design, and/or implementation processes, and many methodologies mix up organizational and information systems modelling constructs. In addition, given the domain to be modelled in multi-agent systems covers a wide range of aspects (such as the agent’s cognitive map, organizational structure, and interactions), it is usually unclear which particular aspect a methodology is intended to model. As a result, it is difficult for people with a specific need in mind to select a suitable methodology for their purpose.

Our survey is intended to address these concerns and provide an up-to-date analysis of the field from a modelling perspective. We are interested in questions such as: what constructs, relations, and models do existing methodologies employ to represent AOIS (in both the analysis and design phases)? In what areas do existing methodologies struggle to provide appropriate models\(^6\)? And where could we search for solutions to these difficulties?

The paper should allow researchers and practitioners to distinguish between the different methodologies, and understand in what environment each methodology could be best employed. It is clear from the work that there is no best A&D technique and each one is focused on a different aspect of the system.

---

\(^6\) By “struggling to provide appropriate models” we refer to certain aspects of AOIS where no sufficient models exist, there is a lack of agreement in the community on the basic constructs, or there fundamental modelling problems impeding advancement.
This paper is different from the surveys done by Yu [1997] and by Iglesias, Garijo and & Gonzalez [1999] and Tveit [2001]. 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 multi-agent implementation architectures and frameworks that are aimed at assisting the developers at the implementation phase. Tveit’s survey is very brief and does not contain in-depth study of the field. It also reviews only a few methodologies. Also, the analysis dimensions that are used in this paper are different and are based on a novel framework. The analysis in this paper is much more comprehensive and detailed. In addition, we include an analysis of the modeling constructs employed for modeling agent systems, and we conclude with a discussion of the major challenges facing this research discipline.

1.5 Scope of the Survey

Although analysis and design (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 behaviour 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 behaviour (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 behaviour of the individual
agent (e.g., agent’s knowledge, goals, plans, and actions) and those of the agent society (e.g.,
authority and acquaintance hierarchies, and coordination and cooperation mechanisms). Thus we
exclude object-oriented techniques from this survey\(^8\).

In this paper, we review purely agent-oriented analysis and design methodologies. Frameworks
that are not strictly analysis and design methodologies (such as agent-oriented development
frameworks and agent-oriented programming languages) are not included in the survey\(^9\).
However, they are discussed in Appendix B, where we review works in relevant neighboring
fields.

1.5 Outline

The paper proceeds as follows. Chapter two lists the works included in the survey and provides
some background for each. Chapter three analyzes to what extent each methodology supports
the analysis and design processes. Chapter four studies the constructs and relations employed by
each methodology and the extent to which they cover the features of multi-agent systems.
Chapter five identifies major challenges and points to possible future directions. Finally,
Chapter six concludes the survey. Appendix A presents a description of each methodology and
Appendix B reviews relevant neighboring fields.

\(^8\) Methodologies and architectures 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

\(^9\) Examples of agent frameworks and implementation tools that are not strictly A&D methodologies are Concurrent
METATEM [Fisher 1996, Fisher 1999a, Fisher 1999b], AOP [Shoham 1993], and SIM_AGENT [Sloman 1999,
Sloman & Logan 1999]. In Appendix B we discuss the relation between these frameworks and A&D methodologies.
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].
2. Methodologies Surveyed

The field of analysis and design of AOIS is still young and it is quite a long way before standard techniques, such as UML in the object-oriented world, would emerge. This research area is fragmented and no one approach is considered a standard. We identified nineteen methodologies, as listed in table one\(^{10}\). Due to space limitations, we leave the description of each methodology for Appendix A.

Below is the list of methodologies included in the survey:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Institute</th>
<th>Methodology</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bradshaw et al</td>
<td>Boeing</td>
<td>KaOS</td>
<td>Bradshaw et al. 1997</td>
</tr>
<tr>
<td>Burmeister</td>
<td>Daimler-Benz</td>
<td>Agent-Oriented Analysis and Design (AOAD)</td>
<td>Burmeister 1996</td>
</tr>
<tr>
<td>Begenti &amp; Poggi</td>
<td>Universita degli Studi di Parma, Italy</td>
<td>B&amp;P</td>
<td>Bergenti &amp; Poggi 2000</td>
</tr>
<tr>
<td>Collinot, Drogoul &amp; Benhamou</td>
<td>University of Paris</td>
<td>Cassiopeia</td>
<td>Collinot et al. 1996</td>
</tr>
<tr>
<td>Elammari &amp; Lalonde</td>
<td>Carleton University, Ottawa, Canada</td>
<td>High Level / Intermediate Models</td>
<td>Elammari and Lalonde 1999</td>
</tr>
<tr>
<td>Glaser</td>
<td>Loria, France</td>
<td>CoMoMAS</td>
<td>Glaser 1996</td>
</tr>
<tr>
<td>Iglesias, Garijo, Gonzalez and Velasco</td>
<td>Universidad de Valladolid, Spain</td>
<td>MAS-CommonKADS</td>
<td>Iglesias et al. 1996</td>
</tr>
<tr>
<td>Kendall, Malkoun &amp; Jiang</td>
<td>Royal Melbourne Institute of Technology</td>
<td>AO Methodology for Enterprise Modelling</td>
<td>Kendall et al. 1996</td>
</tr>
</tbody>
</table>

\(^{10}\) An additional agent-oriented analysis and design methodology, Tropos [Mylopoulos et al. 2000], was not included in this survey. Tropos is a requirement driven methodology, which is based on the I* architecture [Yu 1995]. A description of Tropos is scheduled to appear in Information Systems, Elsevier, Amsterdam, The Netherlands, 2002.
<table>
<thead>
<tr>
<th>Methodology</th>
<th>Organization</th>
<th>Methodology Details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Odell, Parunak and Bauer</td>
<td>James Odell Associates, Siemens, &amp; others</td>
<td>AUML</td>
<td>Odell et al. 2000</td>
</tr>
<tr>
<td>Moulin and Brassard</td>
<td>Laval University, Quebec, Canada</td>
<td>MASB</td>
<td>Moulin and Brassard 1996</td>
</tr>
<tr>
<td>Pont and Moreale</td>
<td>Leicester University, UK</td>
<td>Integrated Software Engineering (ISE)</td>
<td>Pont and Moreale 1996</td>
</tr>
<tr>
<td>Wooldridge, Jennings &amp; Kinny</td>
<td>Queen Mary and Westfield College, London &amp; University of Melbourne</td>
<td>Gaia</td>
<td>Wooldridge, Jennings and Kinny 1999, 2000</td>
</tr>
<tr>
<td>Yu, Du Bois, Dubois and Mylopoulos</td>
<td>University of Toronto &amp; University of Namur, Belgium</td>
<td>I* and ALBERT</td>
<td>Yu et al. 1995, Du Bois 1995</td>
</tr>
<tr>
<td>Caire, Leal, Chainho, Evans, Garijo, Gomez, Pavon, Kearney, Stark, and Massonet</td>
<td>EURESCOM (European institute)</td>
<td>MESSAGE/UML</td>
<td>Caire et al. 2001</td>
</tr>
<tr>
<td>Gervais and Muscutariu</td>
<td>Laboratoire d'Informatique de Paris</td>
<td>ODAC (Open Distributed Application Construction)</td>
<td>Gervais and Muscutariu 2001</td>
</tr>
<tr>
<td>Yu and Schmid</td>
<td>University of St. Gallen, Switzerland</td>
<td>Agent-Oriented Role-Based Workflow Modeling (AORBWM)</td>
<td>Yu and Schmid 1999</td>
</tr>
</tbody>
</table>

Table 1: methodologies included in the survey

As can be seen from the table, most methodologies are developed in academic institutions, with a few developed by business organizations. Another interesting finding is the dominance of European and Australian based techniques.
The methodologies employ as a theoretical foundations either object-oriented (OO) methodologies (the vast majority) or Knowledge Engineering\(^{11}\) (KE) frameworks (with one exceptions). Object-oriented methodologies provide a solid foundation for agent-oriented modeling, since the OO modeling field is mature and these two approaches have many similarities (see Section 1.3). The limitation of OO as a theoretical foundation is that OO methodologies do not provide constructs and models for representing the agent’s mental states and for modeling rich social interactions. KE frameworks are more general architectures and are especially suitable for modeling the organizational environment. Their limitation is that they are not tailored for analysis and design of information systems and lack proper support for these processes. These methodologies, similar to OO techniques, are not designed to support agent’s social interaction.

In the table above, “High Level / Intermediate Models”, CoMoMAS, and MAS-CommonKADS are based on Knowledge Engineering foundations, AORBWM borrows from business process modelling, and the remaining fifteen methodologies are based on object oriented software engineering.

\(^{11}\) Knowledge Engineering is the process of domain knowledge acquisition, knowledge representation, validation and verification, and the construction of a computerized knowledge base. Domain knowledge may refer to an agent’s mind (as in the case of expert systems) or to the objective reality.
3. The Supported Software Engineering Phases

An analysis and design (A&D) methodology is used to guide the development of software systems. Yet, from reviewing works in the field, it is often difficult to identify the software development phases that a particular methodology is supposed to support. Many methodologies use the same constructs and models for representing both the business environment and the information system, and do not make the distinction between analysis and design clear. An important distinction between the various methodologies is the software engineering (SE) phase that they support [Treur 1999a, Mylopoulos 1999]. In this chapter we will map each methodology accordingly and indicate whether or not it differentiates clearly between the phases.

Although various A&D 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., leaving 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].

At the heart of the analysis and design process, there have to be at least two models – the conceptual model and the design model. A&D methodologies include these two models, as well as descriptive and procedural information on how to use the models. A model includes a set of constructs, and relationships between the constructs. The conceptual model includes constructs and relations that capture the static and dynamic nature of the organization and its environment. It should describe the various entities inside the organization and the ones external to it, the different structures and hierarchies in the organizations as well as the business process. The

12 Although in many cases constructs such as ‘agent’ or ‘object’ could be employed at both analysis and design, these constructs represent different entities at the different SE phases – at the analysis they represent real life entities, while at the design they represent elements of the information system. It is important that methodologies make this distinction clear and provide distinct models for modelling the organization and the information system.
design model, on the other hand, includes constructs and relations that describe the structure and behaviour of the information system. The models should describe the multi-agent system as a whole, as well as the internal components of the individual software agent.

We define each of the software engineering (SE) phases below, and then map the surveyed methodologies according to the phases they support.

- **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 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 be realized 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 behaviour of the information system [Coltell and Chalmeta 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.

In Table 2 the cells marked in gray indicate the SE phases that a particular methodology supports.

In the table below we also illustrate whether a methodology clearly distinguishes between the analysis and design phases and whether it provides distinct constructs for modeling the
organizational and modeling the information system. We use a thick separating line (▏) to indicate that the phases are clearly distinguished.

Following is the mapping of the methodologies:

<table>
<thead>
<tr>
<th>Methodology</th>
<th>The supported SE phase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analysis</td>
</tr>
<tr>
<td>KaOS</td>
<td></td>
</tr>
<tr>
<td>AOAD</td>
<td></td>
</tr>
<tr>
<td>B&amp;P</td>
<td></td>
</tr>
<tr>
<td>Cassiopeia</td>
<td></td>
</tr>
<tr>
<td>MaSE</td>
<td></td>
</tr>
<tr>
<td>High Level / Intermediate Models</td>
<td></td>
</tr>
<tr>
<td>CoMoMAS</td>
<td>▏</td>
</tr>
<tr>
<td>MAS-CommonKADS</td>
<td>▏</td>
</tr>
<tr>
<td>AO Methodology for Enterprise Modeling</td>
<td>▏</td>
</tr>
<tr>
<td>AAII</td>
<td></td>
</tr>
<tr>
<td>AUML</td>
<td></td>
</tr>
<tr>
<td>MASB</td>
<td>▏</td>
</tr>
<tr>
<td>ISE</td>
<td>▏</td>
</tr>
<tr>
<td>AOR</td>
<td>▏</td>
</tr>
<tr>
<td>Gaia</td>
<td>▏</td>
</tr>
<tr>
<td>I* and ALBERT</td>
<td>▏</td>
</tr>
<tr>
<td>MESSAGE/UML</td>
<td>▏</td>
</tr>
<tr>
<td>ODAC</td>
<td>▏</td>
</tr>
<tr>
<td>AORBWM</td>
<td>▏</td>
</tr>
</tbody>
</table>

Table 2 – The supported SE phase for A&D methodologies
As shown in the table, only 6 out of the 19 methodologies make clear distinctions between the different SE phases. One possible explanation to this is that many methodologies were originally developed to design systems assuming the organizational context and requirements are known. They were later extended to model systems requirements without considering the different representations needed in the two phases. This might also explain why 5 of the design methodologies do not provide constructs and models to capture the organizational environment (in the analysis phase).

Although the focus of this paper is on analysis and design, and we excluded the methodologies aimed to support implementation, it is interesting to note that two methodologies (CoMoMAS and ODAC) further extended the framework to support implementation. We believe that this is an indication of the maturity of the field.

It is also interesting to point out one methodology (AOR) is strictly for supporting analysis. We believe that strict analysis framework has limited utility for developing information systems and eventually needs to be extended to support systems design.

In the following chapter, we will further analyze the methodologies according to how much multi-agent systems features they cover. We will then show how the analysis can be combined with Table 2 in this chapter to highlight the uniqueness of each methodology.
4. Coverage

The second dimension for mapping the methodologies is the degree to which the methodology provides constructs and models for covering the wide spectrum of multi-agent systems features. Since no one has established an agreed-upon set of constructs for this purpose and no applicable ontology has been identified, we offer a framework for studying representational issues of agent systems. We divide the multi-agent ‘representational space’ into four sub-domains, and then for each of these sub-domains we identify a set of construct clusters that are used (e.g., planning, belief and knowledge, and communication). We take a bottom-up approach to study the representational components of existing agent-oriented A&D methodologies. We then map existing methodologies according to their coverage of the construct clusters, in each of the representation sub-domains. Lastly, we discuss the finding from our coverage analysis and try to provide some insights.

4.1 Multi-Agent Taxonomy

To study the representation models employed by the methodologies, we first need to agree on a set of appropriate constructs. This is a critical issue, since no such set of constructs has yet been identified. 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 were 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.”

There is a wide agreement on the need for a methodology that will overcome these limitations [Luck et al. 1997]. The field is relatively young and it would take quite some time before an acceptable standard, such as UML in the object oriented-world, would emerge [Treur 1999a, Treur 1999b].

One of the organizations that have been working toward realizing this goal 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 Franco Zambonelli\textsuperscript{13}, is trying to draw a roadmap leading to standards in agent-oriented analysis and design. In a series of meetings [Treur 1999a, Treur 1999b, Zambonelli 2001, Luck et al. 2001, AgentLink 2001] they have defined short term, medium term, and long term objectives for developing standards in multi-agent Software Engineering. This paper is complementary to the work of AgentLink and could serve as a basis for the next step in the SIG’s mission.

If no standard exists, then how can we come up with a set of constructs and relations necessary for representing the complexities of agent systems? One possible approach is a top-down analysis, which is based on an ontology\textsuperscript{14} – a formalized conceptualization of a domain. Wand and Weber [1990, 1993] explored in a series of works the use of Bunge’s ontology for guiding the development of object-oriented A&D methodologies. To date no such attempt has been made in the agent world, and no ontology has been identified as a suitable candidate for modelling agent systems. Clearly we need an alternative approach for the purpose of this paper.

Below we propose an alternative approach for studying the representation of multi-agent systems. The proposed framework is based on two complementing types of analysis. First we conduct a top-down study by reviewing the literature and identifying several important agent-systems sub-domains that need to be represented. Next we apply a bottom-up approach and study the existing agent-oriented A&D methodologies.

Two ways of categorising multi-agent systems are common in the literature: (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 \textit{et al.} 1995, Brazier \textit{et al.} 1997b, Sloman and Logan 1999, Ciancarini \textit{et al.} 1999], and (b) the dynamic aspects (behaviour) 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

\[\text{-----------------------------}\]

\textsuperscript{13} Who took over from Jan Treur

\textsuperscript{14} Ontology is the study of the things that exist in reality, and has been traditionally been studied within Philosophy.
the multi-agent system, or what we term ‘a representation sub-domain’. Although we have not
encountered such a taxonomy before, we believe that this is useful in thinking about agent
systems and can also be used for people in the field to situate their work.

<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>

Figure 1 – taxonomy for multi-agent systems

The top left cell refers to the *Individual Agent / Static structure* sub-domain. Models, constructs
and relation in this sub-domain describe how the internal (cognitive and emotional) map of the
agent is structured. The bottom left cell refers to the *Individual Agent / Dynamics* sub-domain.
Models and constructs in this sub-domain deal with the functionality of the agents: the evolution
of the agent’s knowledge and goals, planning, and learning. The top right cell refers to the
*Social System / Static Structure* sub-domain. Models and constructs in this sub-domain describe
the structure within the agent society (e.g., organizational structure, role hierarchy, and goal
hierarchy and the task-resource, task-skills, and roles-tasks relations), and the normative
component of the social system, such as permissions, rights, and norms. The bottom right cell
refers to the *Social System / Dynamics* sub-domain. Models in this sub-domain represent the
interactions and communications between agents, and the way in which agents coordinate their
activities, cooperate, and compete. Together these four sub-domain compose the
‘representational space’ of agent systems\(^{15}\), which we will use to analyze the coverage of each
methodology later.

---

\(^{15}\) Additional important aspects of analysis and design that could be thought of as domains, yet not included in the
analysis, 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. Although these concepts (time and uncertainty) 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.
4.2 Concept Clusters

Below we describe the concept clusters we have identified in each sub-domain. These clusters were constructed by first analysing the constructs and relations employed by the methodologies surveyed in this paper, mapped them onto the sub-domains, and then identified a set of concept classes or clusters in each. In addition, we use the KaOS methodology to illustrate the coverage in each sub-domains.

**Individual Agent / Static Structure:**

The concept classes listed below are used to describe the internal structure of the agents:

- **Perception** – what events in the world the agent is capable of perceiving
- **Knowledge** – the agent’s knowledge and **beliefs** about the world, himself and other agents
- **Transparency** – what parts of the agent’s knowledge he is willing to **expose** to other agents
- **Emotions** – the agent’s emotional state (rarely used)
- **Goals** – a list of agent purposes, goals, **preferences**, or **desired** states (possible with value or utility associated with each)
- **Resources** – a set of resources, skills or capabilities the agent has access to
- **Actions** and **communications** – a set of actions the agent is able to perform. Communications are a specific type of action, which involves information exchange with other agents
- **Plans** – a set of possible plans for achieving the goals. Each plan includes a pre-defined set of actions and the required resources.
- **Intentions** and **commitments** – triggered plans (also referred to as commitments to perform an action)

We will use the KaOS methodology [Bradshaw et al. 1997] to explain this representational sub-domain. KaOS includes the following attributes of the agent’s cognitive model: 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, transparency and perception constraints, and

---

domain. Thus, we did not perform an analysis of the methodologies according to the **Time** and **Uncertainty** dimensions.
constructs that relate to the emotional state of the agent are not included in this model. The coverage of KaOS for this sub-domain, thus, is medium.

Individual Agent / Dynamic:

The concept classes listed below are employed to represent the events that trigger changes in the agent’s internal structure:

- **Acting** – events that trigger an agent to (re)act
- **Reasoning** (for rational agents) – a mechanism for choosing the optimal plan for obtaining a goal based on the possible set of actions, the costs associated with each, and the utility associated with the goals
- Activating a plan – the events that trigger a plan (and turn it into an intent for actions)
- Activating an intention – the events that trigger an intention, causing the agent to (pro)act
- **Learning** and **evolution** – the events that trigger changes to the agent’s knowledge, goals, or plans

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. During their lives agents receive, 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 the basis of their intentions, without considering the implications of their actions and other agents’ actions on the environment. Thus, KaOS’s coverage here is medium.

Social System / Static Structure

The system’s structure could be represented with the following set of construct classes:

- **Resources** – the existing resources, skills and capabilities in the system and their organization (i.e., resource hierarchy)
- **Goals** hierarchy – goals and their (de)composition. May also include a description of the pre and post conditions for the goals
- **Actions / Tasks / Services** – the possible actions in the system and their organization (i.e. task decomposition or activities hierarchy)
- Agent **classes** – describing the different classes of agents in the system and (hierarchical) relations between classes
- **Roles** – a hierarchy of agent’s roles and the agent classes associated with each role.
- **Permissions** – the rights to use resources (for agent classes or roles)
- **Responsibilities** – the functionality (i.e. actions) and goals associated with roles or agent classes
- **Acquaintance** (or **friendship**) network – a network of relations between agent classes or roles that define the group of agents an agent will be interacting with (i.e., communicating and sharing information)
- **Authority** hierarchy – the authority and **control** structure in the system (between agent classes or roles), which define for each agent his subordinates and superordinates.
- **Global constraints** – transparency and perception constraints (see “Individual Agent / Static Structure”) on agent classes or roles
- **Dependency** relations – system level constraints describing dependencies between agent classes or roles (dependencies for using resources, performing tasks, or achieving a goal)

Using the same example given earlier, 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. KaOS coverage of this sub-domain is, therefore, low.

**Social System / Dynamics**

The system’s behaviour is emerging through the individuals agent’s behaviour, and system level constraint (on the behaviour) are described under “Social System / Static Structure”. Thus under “Social System / Dynamics” we include the description of the possible interactions between agents, and agents’ cooperation.

- **Messages** – possible messages, including the sender and the receiver, and the contents of the message (many times defined in terms of Speech Acts)
- **Conversations** – conversations and **communication protocols** define the sequence of messages that make up a conversation
- **Coordination** – coordination and **conflict resolution** mechanisms that are necessary to enable agents **cooperation** (could be based on responsibilities, acquaintance and authority of agents)
- **Mobility** (for mobile agents) – agents’ **location** over time, and the event that trigger location change

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. Thus, KaOS’s coverage for the “Social System / Dynamics” sub-domain is high.

4.3 Mapping Methodologies by Coverage

Above we described a framework for studying the constructs and relations used for agent-systems. Following we proceed to map the methodologies according to that framework. We evaluate each methodology on the extent to which it covers the concept clusters described above. We define three categories of coverage:

- High – the methodology provides a comprehensive set of constructs, and the models include complex hierarchies and relations. For example complex models of agent societies will include hierarchies of roles, authorities, responsibilities and interactions between those hierarchies. This enables a structured modelling process, yet rather difficult and time-consuming.

- Medium – the methodology provides a pre-defined set of constructs and relations, but the number of supported constructs is rather limited. Some methodologies allow adding constructs on top of the restrictive pre-defined set. Medium coverage methodologies are usually designed for a specific application domain, and allow for a relatively easy and straight-forward analysis and design process. The best examples of such a model are methodologies based on BDI\textsuperscript{16} architecture [Kinny, Georgeff and Rao 1996, Georgeff

---

\textsuperscript{16} 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.
and Rao 1995]. The set of constructs available for modelling the agent’s internal structure are very limited, which enables the agent to formulate a tractable\textsuperscript{17} plan.

- Low (but expandable) – the methodology does \textit{not} provide a pre-defined set of constructs, but supports the adding of constructs and relations. These methodologies are very general and could be applied to different application domains. An example is Gaia’s [Wooldridge, Jennings and Kinny 2000] model of the agent’s internal structure – in order not to commit to specific constructs and leave much flexibility to the designer, Gaia does not pre-specify constructs. This type of methodology enables an undirected and flexible analysis and design process, which is more appropriate for experienced designers.

In table 3 we map each methodology according to its coverage, as explained above, in each of the representation sub-domains. On top of the three categories defined earlier: high (a pre-defined and comprehensive set of constructs and relations), medium (a pre-defined, but limited, set of constructs and relations), and low but expandable (no pre-defined constructs, ability to define constructs), we add one more category – \textit{none}. We use this additional category when the methodology does not provide any constructs or models for representing a specific sub-category. Further information about the constructs, relations and models the methodologies employs for mapping each sub-domain is included in Appendix A.

The following legend is used in the table:

\begin{tabular}{c c c c}
\hline
\textbf{High} & \textbf{Medium} & \textbf{Low} & \textbf{None} \\
\hline
\end{tabular}

\textsuperscript{17} Tractable models are models that can be solved by computers in a finite time and using finite storage spaces.
<table>
<thead>
<tr>
<th>Methodology</th>
<th>Individual level</th>
<th>System Level</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure</td>
<td>Dynamics</td>
<td>Structure</td>
</tr>
<tr>
<td>KaOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B&amp;P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cassiopeia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High Level / Intermediate Models</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoMoMAS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAS-CommonKADS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AO Method. for Enterprise Model.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAII</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUML</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I* and ALBERT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSAGE/UML</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AORBWM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 – Coverage of constructs clusters in the representational sub-domains.
The above table provides a summarize view of all 19 methodologies on one page. This could be of great assistance to practitioners in choosing a methodology for a specific application domain. It also provides insights of the state of the field. We will elaborate both of them below.

This comparison dimension seems to be effective - only a few methodologies present a similar pattern. In specific cases where methodologies provide similar patterns, we can combine the first analysis dimension (the supported SE phase) to make the differences clear. For example, AUML and B&P present similar patterns in the table above, but when consulting table 2 we notice that AUML could be employed to support both the analysis and design phases, while B&P support the only the design phase.

How can these maps be used to help the system developer in choosing an appropriate methodology? We illustrate this with two examples. Assume that a researcher is studying human social interaction and is looking to develop an agent-based simulation. The researcher is especially interested in tools to support the analysis and modeling of a group of people’s interaction. The methodology best suited for his needs would support the analysis phases (1st analysis dimension) and provide high coverage models for representing societal level dynamics. “High Level \ Intermediate Models” seems to be a good choice for that specific need.

A second example would be a company looking to automate their manufacturing processes by utilizing agent technology. The company is looking to develop a scheduling program based on a market driven agent interaction. Each part and each machine on the manufacturing floor will be represented by agents. An appropriate methodology for this problem would support the design phase (most critical), to a lesser degree the analysis phase, and possibly the implementation phase. The software agents in this system will need to formulate a plan and to collaborate with each other. Thus is it essential that the supporting analysis and design methodology employed will provide tractable models for describing the dynamics of the individual agents (specifically planning) and of the agent society (specifically agent communication and conflict resolution mechanisms). “I* and Albert” seems to be a good choice of methodology for these purposes.

On a more general level, we obtain the following insights from the table:
• There is an agreement that agent-oriented A&D methodologies should put emphasis on system/societal aspects, and many of them cover these aspects well. This could be due to the availability of some commonly accepted techniques or approaches such as communication protocols.

• There are two different viewpoints on covering the individual agent aspect. Some methodologies leave the individual aspects open-ended while others provide a very comprehensive set of constructs to model them. This indicates a lack of convergence on the necessary constructs for modeling an agent’s internal structure and behaviour. We will revisit this point again in Chapter 6 when we discuss critical issues and future challenges in the field.

• It is interesting to note that approaches based on Knowledge Engineering as a theoretical foundation (High Level / Intermediate Models, CoMoMAS, and MAS-CommonKADS) seem to include the most extensive set of constructs and relations and the most comprehensive models. We conjecture that it is difficult to extend the object-oriented concepts to the modeling of agents, and that the Knowledge Engineering field provides a good foundation to overcome this challenge.

Up to now we have described the process of agent-oriented systems analysis and design and reviewed existing methodologies. We have highlighted some of the differences between the existing approaches and tried to provide guidelines on when (i.e., in what application domain) each methodology would be more appropriate. This part is relevant to both practitioners and academics. In the following chapter, we will discuss critical issues and major challenges to this field of research. The discussion is aimed at the group of researchers working on the development of analysis and/or design methodologies for agent-oriented systems.
5. Critical Issues and Possible Future Directions

Below we discuss critical issues for analyzing and/or designing agent systems. We will discuss both general issues of Software Engineering and issues related to modeling agent systems.

Following we describe three fundamental issues of software engineering. Although these issues are not unique to analysis and design of AOIS and we have already discussed them in Chapter 3, we feel it is important to highlight them since many existing methodologies struggle with these issues.

- **Covering and distinguishing the analysis and design phases** – many existing methodologies struggle with supporting both the analysis and design phases, and making the distinction between the phases clear. We believe that it is essential to provide tools for capturing the complexity of the organizational environment (conceptual models) and for designing heterogeneous and complex multi-agent system (design models), and that these phase should be clearly distinguished by providing distinct representation models for each phase.

- **Guiding the analysis and design process** – many methodologies do not provide explanations and guidelines on using the diagrams and models and on how to proceed from the conceptual model to the design model [Wand and Woo 1999, Yu et al. 1995]. We believe that it is important to guide the systems analyst and designer through the A&D process (e.g., provide clear guidelines for deciding what entities should or should not be represented by agents and what types of agents they should be, and for relating agents in the analysis models to agents in the design).

- **Approaching design from requirements** – the software engineering process is not linear, and there is much interaction between the SE phases: analysis, design, and implementation. Yet, it is important that the process be driven by the requirements. This is especially crucial in large-scale systems development when different teams work on different aspects or phases of the development. Several methodologies either do not support the analysis phase or allow describing system specifications without describing first the requirements.
We believe that the issues described above are critical and it is essential for agent-oriented A&D methodologies to address them.

From a modelling perspective, the major obstacle to progress in the field is the lack of convergence on the constructs needed to represent agent systems. We discussed this problem earlier, and the difficulty of defining such a set of constructs. In section 4.2, we tried to make a contribution in that direction, by conducting a bottom-up analysis and identifying the constructs employed by existing agent-oriented methodologies.

Below we list some areas where the lack of agreement on a set of constructs is especially acute. We also point to some possible future directions, based on insights we have gained from reviewing neighbouring fields\footnote{These neighbouring fields are agent-oriented programming languages and control architectures, agent-oriented simulation and automatic prototype generation frameworks, DAI agent-oriented problem solving frameworks, and Computational Organizational Theory (COT). In Appendix B we describe these research fields and explain how ideas from these areas could be relevant to the modelling of agent systems.}

- Modelling the agent’s internal (or cognitive) structure – there is no theory to guide the selections of constructs and models for representing the agent’s knowledge, goals, plans and learning mechanisms. Existing approaches are either guided by implementation issues (for instance BDI architectures, which are designed for tractability) or by common sense.

Possible theoretical foundations could be Cognitive Psychology and Cognitive Anthropology, specifically the design of Cognitive Architectures\footnote{[University of Michigan 1999] includes a comprehensive study of the theory in this research field, as well as description of specific architectures.}. 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]. This is specifically relevant to the design of the model of the agent’s cognitive (and emotional) structure. Many methodologies borrow from psychology and use in their psychological constructs such as ‘beliefs’, ‘intentions’, and ‘attitudes’, but an adoption of a more comprehensive psychological framework is lacking. Ad-hoc approaches that are based on common sense provide a good
starting point, but ultimately we should build on solid theories of human behaviour. These theories provide a comprehensive and detailed model of human cognition, emotion, and action. The work of [Carley and Gasser 1999] tries to link the agent’s cognitive architecture to the agent’s capabilities and the tasks he needs to perform. This work is based on an earlier work by Carley and Newell [Carley and Newell 1994], and is an excellent source for understanding the complexity associated with modelling the agent’s cognitive architecture.

An example of a framework from a neighbouring field (Agent-Oriented Simulation and Automatic Prototype Generation Frameworks; see Appendix B) that builds extensively on knowledge from psychology and formal social theories is STEAM [Tambe 1997].

- Modelling agent society’s structure - agents represent a new abstraction, more complex than objects, and in many ways software agents resemble human agents. To capture the structure of this complex system constructs, such as roles, authorities and rights are used. Agent-oriented 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]. The social 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).

To date agent-oriented methodologies employ social concepts (for example acquaintance networks and authority hierarchy), however none of these methodologies builds on solid sociological theories. Current use of social constructs is in an ad-hoc manner, with no reference to models of human social structure. We believe that we should draw on formal social theories and employ knowledge from Anthropology and Organizational sciences when designing an agent-oriented A&D methodology.

An example of a framework from a neighbouring field (Computational Organizational Theory; see Appendix B) that provides comprehensive models for describing the agent’s society structure is TOVE [Fox et al. 1998].

- Modelling the agent society’s behaviour – if agent societies are represented in terms of human societies, than constructs and models for representing the system level interactions
should be based on models of human social behaviour. If agents are to compete, cooperate, 
share resources, and coordinate their activities, than abstraction for representing these notions 
are required.

As with modelling of agent’s system structure, existing methodologies define constructs in 
an ad-hoc manner. We believe that theoretical foundation are necessary for modelling the 
behaviour of agent societies, and, again, we point to the fields of Organizational Science and 
Anthropology as possible candidates.

- Developing precise relations between the organizational structures and local specifications – 
modes of the system should be related to models of the individual agent’s structure and 
behaviour. Hierarchies at the organizational level could be used to restrict agent behaviour, 
for example the friendship network could specify which other agents the agent cooperates with.

- Enabling agent interactions in an open environment – as open environments become ever 
more popular, methodologies should support development for borderless heterogeneous 
systems. To allow agents to interact effectively and cooperate in an open and heterogeneous 
environments two challenges have to be addressed: (a) enabling semantic interoperability, so 
agents designed by different groups could ‘understand’ each other, and (b) enabling 
interoperability through coordination languages and conflict resolution mechanism, so agents 
with conflicting goals could arrive at an action. Both of these challenges are difficult and no complete 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 et al. 1999]).

Other discussions of challenges in developing agent-oriented software engineering 
methodologies can be found in [Brazier et al. 1999] and in [Yu 1997].
6. 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.

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. There is no existing methodology that can support both analysis and design, and effectively cover all constructs necessary for modelling agent systems.

Although there are some fundamental challenges facing this research community and we are yet far from reaching a well-accepted standard for modeling agent systems, several approaches are gaining the leading position, based on the functionality of the methodologies and the reputation of the research team. We identify four such approaches:

- Methodologies based on Knowledge Engineering, and specifically on CommonKADS (part of the European Community ESPIRIT program). The two examples we mentioned in this paper are CoMoMAS [Glaser 1996] and MAS-CommonKADS [Iglesias et al. 1996]. The strengths of these approaches are in modeling the dynamic aspects of AOIS.

- Methodologies based on the BDI architecture. Example are AAII [Kinny, Georgeff and Rao 1996, Georgeff and Rao 1995] and “Agent Oriented for Enterprise Modeling” [Kendall et al. 1996]. The power of these methodologies is in modeling agent’s planning capabilities and in providing a tractable mechanism for computing plans.

- Extensions to UML. The most notable examples are AUML [Odell et al. 2000], MESSAGE/UML [Caire et al. 2001], B&P [Bergenti & Poggi 2000], and MaSE [DeLoach 1999, Wood & DeLoach 2000]. These methodologies build on existing object-oriented standards and extend them mainly to model agents’ interactions.

- Gaia – a methodology by Wooldridge, Jennings and Kinny [Wooldridge, Jennings and Kinny 1999, 2000], which makes the analysis and design process clear and structured, and provides rich models to capture the structure of agent societies.
There are several contributions of this work. First, this is by far the most comprehensive survey of the field. The paper provides numerous references to works in the field of agent-oriented analysis and design, and in related fields. Thus it could serve as a valuable resource for both practitioners and academics.

Second, we define the analysis, design and implementation phases, and present a mapping of agent-oriented analysis and design methodologies accordingly. In doing so, we identify methodologies that follow good software engineering practices, making a clear distinction between the models aimed for capturing system’s requirements and models for representing the information system.

Third, we develop a framework for studying the modelling of agent systems. We decompose the agents ‘representational space’ into four sub-domains, and through a bottom-up analysis, identify important clusters of constructs for each sub-domain. In addition, we provide a mapping of each methodology according to its coverage of the construct clusters. The framework we develop could be used to situate and differentiate existing works. Furthermore, researchers in the field can use the set of identified construct clusters to guide the discussions on the convergence of a set of constructs.

Finally, we identify critical issues facing researchers in this field and point to possible solutions.
References


[Franklin and Graesser 1996] Franklin, S., and Graesser, A., Is it an agent, or just a program?: a taxonomy for autonomous agents. in Proceeding of the Third International Workshop on


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


Appendix A - Description of the Methodologies

Appendix A includes a description of each one of the methodologies described in Chapter 2. Each description includes an overview of the methodology, it’s aim, and the theoretical and logical foundation it employs.

We employ the framework described in chapter four for decomposing the representation domain of multi-agent systems into four sub-categories. We evaluate the modeling capacity of each methodology in each of the four representational sub-categories. The summarized result of this evaluation process is presented in table 3 in the main body of this document.

Following is a description and an evaluation of the agent-oriented frameworks. Since experiencing with all the methodologies would require a lifetime of effort, we evaluated the methodologies based solely on the description provided in the referenced articles. It is possible that certain features that appear to be missing actually exist in the methodology and were simply left out of the paper. It is also possible that in the most recent version of the work, the methodologies have been updated or enhanced.
KaOS; Bradshaw et al.

KaOS is an architecture for designing agent systems that focuses on the communications among agents, and hence can be considered as an open agent communication meta-architecture [Bradshaw et. al 1997]. It aims at providing an infrastructure for implementing and integrating diverse types of agent-oriented systems. The KaOS architecture is neutral with respect to: hardware platforms, operating systems, transport protocols, programming languages, and the type of communication primitives used. The design of the system is based on Object-Oriented Programming approaches where agents are treated as “objects with intentions”. A hierarchy of agent classes exists, and each agent is an instantiation of a specific class. Agents inherit attributes based on the class hierarchy.

Individual Agent / Static structure
KaOS includes a relatively simple model of the agent’s cognitive map, which includes the following attributes: 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. There is no exact description of how these attributes are related to each other nor on how the cognitive map leads to the agent’s actions.

Individual Agent / Dynamics
The agent’s dynamics model include 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) [Bradshaw et. al 1997 p. 384]. During their lives agents read, process and send information continuously, and according to the information they receive they may update their internal structure. 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 in the environment.
Social System / Static Structure

The definition of the social structure within 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, authority networks, etc.).

Following is a short description of each one of the agent’s types used in the model:

- KaOS agent: is defined according to the attributes specified above, and is the main component of any agent system. This is the agent that actually performs the work the system is designed to do.
- Mediation Agents: provides interface between KaOS agents and external entities.
- Proxy Agent: is a special case of the Mediation Agent, which provides interface between two KaOS agent domains.
- Domain Manager: controls the exists and entries of agents into the domain (according to pre-defined guidelines), registers all the agents in the domain and keeps a list of their current addresses.
- Matchmaker: is a special case of the Mediation Agent, and is responsible for registering the services each one of the agents provides. This agent receives requests for service from the KaOS agents and matches them with the agent that provides the requested service.

In this architecture only a very simple type of broker is used as a middle agent within the KaOS agent domain – the Matchmaker. Other, more complex middle agents, such as subcontractors, are not available in this architecture.

Social System / Dynamics

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. Each one of these sequences is pre-defined, and hence when designing a KaOS architecture the designer should elicit all 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.
Gaia; Wooldridge, Jennings & Kinny

Gaia is a methodology for agent-oriented analysis and design [Wooldridge, Jennings and Kinny 1999, Wooldridge, Jennings and Kinny 2000], and it could be used as a meta-level model. Gaia includes all the aspects that are important in describing agent societies (individual agent aspects as well as social aspects; static aspects as well as dynamic ones), but only at a high level. The reasoning behind this approach is the desire to leave low-level design and implementation issues as open ended as possible, allowing the designer to choose the architecture and programming language. While this approach has the advantage of being general and implementation independent, it does not provide the designer with all the necessary tools to analyze and design the system.

Gaia makes an important distinction between the analysis (dealing with abstract concepts) and the design (dealing with concrete concepts) processes, and provides several models for to be used at each phase:

The Analysis phase is described using the following models:

- Roles model, which is composed of several other models:
  - Permissions
  - Responsibilities (Safety properties and Liveliness properties)
  - Protocols
- Interactions model

The Design phase is described using:

- Agent model
- Services model
- Acquaintance model

The analysis models should be elicited first, and then the design models could be derived from them, while adding more details. Some approaches to information systems analysis and design claim that the translation process from the analysis phase to the design phase should be automatic and should not provide for any degrees of freedom; Gaia does not provide such direct translation.

Another aspect of the design process, which is missing in Gaia as well as most analysis and design methodologies, are specific directions on how to perform the analysis process,
and how to answer questions such as “what entities in the domain should become agents?” and “what is the goal of each agent?”

Gaia is based on object-oriented concepts, but it adds the notion of virtual organization of agents (somewhat similar to the ideas of COT – Computational Organization Theory).

The analysis and design process is similar to the process of constructing a society of agents, defining the role and capabilities of each individual agent, and the way the society of agents is structured.

**Individual Agent / Static structure**

Gaia provides no support for constructs describing the internal structure of the agent, hence it is not clear what mechanism is used to transform the agents perceptions into actions. This part of the methodology is completely open-ended, and leaves the designer the absolute freedom in deciding what cognitive model to use.

**Individual Agent / Dynamics**

In the *Services Model*, which is defined during the Design process, the functionality of each agent type is specified. This model includes a description of the inputs, outputs, pre-conditions and post-conditions for each service. The services are derived from the protocol model (where the functions and goals of each role are detailed), and the pre and post conditions are derived from the responsibilities of the role.

Gaia does not prescribe an implementation approach for the services.

**Social System / Static Structure**

The *Roles model* is used in the Analysis phase to describe the social structure of the agent society. An agent plays a role (many to many relationship), and each role is associated with *responsibilities* (the agent’s functionality and it’s goals), *permissions* (the right associated with the role, mainly rights to use resources, which allow the agent to realize the responsibilities of the role), and *protocols* (the allowable types of interactions).

The notion of resource used here is very limited and includes only information and knowledge (which the agents have).
The Agent Model is used in the Design phase to specify the hierarchy of agent classes. Inheritance is not part of this model and the model only relates roles to agent classes, where an agent class, or agent type, may include one or more roles. The agent model also describes the number of instantiations of each agent type, i.e., the actual number of agents from each class that will be implemented.

The possible communication links between agents are defined in the Acquaintance Model, which is basically a very simple graph linking agents types.

Social System / Dynamics

The Interactions model (Analysis phase) is used to specify the possible interactions between roles. Each protocol is an institutionalized pattern of interaction, and is very schematic – it specifies only the purpose, initiator, responder, inputs, outputs, and processing of the conversation, but not the exact ordering of messages (which is defined at a later stage).

There is no model that describes the ordering of messages and the types of communication allowed, such as the one available in methodologies that use speech act models.

Gaia does not deal with the complex behaviors that can arise in an agent society, and it provides no tools for expressing notions such as Trust, Fraud, Commitment, and Security.
Australian AI Institute (AAII); Kinny, Georgeff and Rao

The AAII methodology (Australian AI Institute; although not mentioned in the references, this is the name this methodology is mostly referred to) [Kinny, Georgeff and Rao 1996, Georgeff and Rao 1995] is based on the BDI (Beliefs, Desires, Intentions) approach for designing agent-oriented systems. AAII is not an analysis and design methodology, but only a design methodology, as it does not include a conceptual model of the organization and its environment; it includes only concrete entities, which relate to data objects.

This methodology consists of two viewpoints. The external viewpoint describes the social system structure and dynamics. It includes an Agent Model (the static structure of the system) and an Interaction Model (the dynamics of the system). The external viewpoint is independent of any internal structure of the agent (the cognitive model) and independent of the communication mechanism.

The internal viewpoint is composed of three models: the Belief Model, the Goal Model, and the Plan Model. These models specify how an agent perceives the environment and how he chooses his actions based on his perception.

The design of agents in this methodology is restricted to the BDI model. BDI agents are capable of rationalizing about their actions and creating an optimized plan (based on their knowledge).

Individual Agent / Static structure

The cognitive model is specified in the internal viewpoint and it includes the following concepts: Events the agent may perceive, Actions the agent performs, Beliefs he holds, Goals he adopts and Plans, which give rise to his intentions.

The Belief Model specifies the knowledge of the agent. The agent has beliefs about: the environment, his internal state and the actions he is capable of performing. Belief Sets are comprised of the possible beliefs and their properties, and are described using the Belief Set diagrams. The goal Model specifies the goal states: the goals an agent may adopt and the events the agent can respond to. The Plan Model includes a description of the Plan
Set – plans that the agent may possibly employ, and is depicted using the Plan diagram (which is an extension of a state chart).

**Individual Agent / Dynamics**
The functionality of the agent and the way in which the internal state of the agent may evolve over time are not part of the BDI methodology, since the methodology is restricted to one specific architecture – the BDI model. This model describes specifically how all of the concepts that are included in the agent’s cognitive map interact and affect each other and the agent’s planning mechanism. The designer has no freedom in defining this part of the model and hence it is not part of the methodology. This is both the major advantage and disadvantage of this methodology. It commits to a very detailed and clear internal architecture, which saves a lot of time and effort, but at the same time it restricts the designer to adopt the BDI approach.

This model does not include learning.

**Social System / Static Structure**
The structure of the system is specified in the Agent Model, which includes a description of the agent classes and their relations (similar to the Roles concept in most other methodologies) and the agent instances of each class and when they come into existence. These are depicted in the agent class and instances diagram.

There is no explicit description in the methodology of how to design the agent society and what sorts of agents (e.g., middle agents and mediators) will be implemented. Other components of the system structure are included in the Interaction Model, such as the responsibilities of each agent class and the control relationships between classes.

**Social System / Dynamics**
The social activity is described in the Interaction Model, where the syntax and semantics of the interactions is defined.
MaSE – Multi-agent Systems Engineering

MaSE is a methodology and a language for designing agent systems. It was first introduced at the Agent99 conference in Seattle, by Scott DeLoach [DeLoach 1999, Wood & DeLoach 2000].

The methodology includes four steps: Domain Level Design, Agent Level Design, Component Design, and System Design, to be followed in that order. It uses two languages, AgML (Agent Modelling Language), which is a graphical language, and AgDL (Agent Definition Language), to describe the system level behavior and to specify the internal behavior of the agent, respectively. AgML is used in the Domain Level Design and the System Design steps, while AgDL is used in the Agent Level Design and Component Design steps. Each language contains a number of diagrams that describe different aspects of the system.

MaSE is a design driven methodology. It does not include an analysis process, and thus it does not provide tools for capturing the organizational environment. However, it goes beyond the design to support some of the initial implementation process. The methodology is based on object-oriented techniques (mainly OMT and UML), and extends them by adding constructs to capture the specific behavior of multi-agent systems. MaSE was developed to support formal system synthesis and it is based on first order predicate logic.

Below we’ll try to classify the steps and the languages of MaSE according to the individual/system and static/dynamic categories, although MaSE does not follow this categorization.

**Individual Agent / Static structure**

The agent’s internal structure is defined in the third step – Component Design. There is no specific cognitive model that this methodology builds on, and user is free to program any structure he may see fit.
**Individual Agent / Dynamics**

The agent behavior is specified in the second step – Agent Level Design, using the AgDL language. This step builds on the agent conversation, which was identified in the previous step (Domain Level Design), and comprises the following steps: 1. Mapping actions identified in the conversations to internal components, 2. Defining data structures that were identified in the conversations (the data structures represent input and output from the agent), and 3. Defining additional data structures, internal to the agent.

There are no generic planning or scheduling architectures within the methodology, and the user is free to implement any planning algorithm s/he may choose.

**Social System / Static Structure**

The system’s structure and dynamics are specified by the AgML and its diagrams. In the Domain Level Design, the agent types are identified and some of the system’s dynamics are defined (see below). Three diagrams that are part of the AgML are used in the Domain Level Design: Agent Diagram, Communication Hierarchy Diagram, and Communication class Diagrams. The Agent Diagrams describe the agent classes (including services and goals for each class) and their hierarchy.

**Social System / Dynamics**

The system’s dynamics is also specified using the AgML during the Domain Level Design and the System Design steps. The Agent Diagram that is used in the Domain Level Design step describes, in addition to the agent classes, the possible coordination protocols (or conversations) between classes. Each conversation has a class hierarchy, which is specified in the Communication Hierarchy Diagram. This diagram describes the relations between various conversations in the system. The Communication Class Diagrams are a set of finite state machines that define the states of each conversation. MaSE provides a powerful tool for representing the agent’s communication, but it lacks tools for capturing some of the more complex social interactions.
**High Level and Intermediate Models; Elammarri and Lalonde**

Elammarri and Lalonde introduced their agent-oriented methodology in the Agent99 conference at Seattle [Elammarri and Lalonde 1999]. The methodology builds on two processes: High level (the Discovery phase) and Intermediate models (the Definition phase). The Discovery phase deals with describing the organizational processes and workflow, and hence is very similar to our notion of an analysis phase. The authors do not distinguish between analysis and design phases and use the term agent in both the Discovery and Definition phases, but clearly the high-level model is used to capture the behavior of the organizational environment where the system is to be implemented (and hence can be considered a conceptual model), and the intermediate models describe the architecture of the information system (and hence should be considered design models). The analysis in the Discovery phase is done using UCMs (Use-Case Maps), which are very useful for visualizing a workflow and work processes. The Discovery phase provides a description of the scenarios, components, roles, scenario’s pre and post conditions, and component’s responsibilities and constraints.

The Definition phase includes four models: the Internal Agent model, the Relationships model, the Conversation model, and the Contract model. These models are described below. The methodology includes guidelines on how to generate the models and how to answer questions such as “what entities should be represented by agents?” and “how to distinguish between the various agent types?” It also provides guidelines to define interrelations between the models.

The Elammarri and Lalonde methodology is based on existing approaches in Software Engineering, and adjusted them to describe multi-agent systems.

**Individual Agent / Static Structure and Dynamics**

Both the agent’s internal structure and the agent behavior are represented in the Internal Agent Model, which is derived from the high level model. This model includes a description of the agent’s goals (a desired state), pre-conditions (the beliefs that should
hold in order for the goal to be executed), post-conditions (the effect of executing a goal on the agents beliefs), and tasks that are required to fulfill each goal.

Several plans could be used alternatively to realize a goal, and each plan could be specified using a finite state machine.

The agents in this model are not able to make decisions or solve problems nor can they learn from experience.

**Social System / Static Structure**

The Relationship model specifies inter-agent dependencies (using the dependency diagram) and jurisdictional relationships (using the jurisdictional diagram). This model is also derived from the high level model. The dependency diagram describes dependencies between service providers and agent that require services. There are four types of dependencies: goal, task, resource and negotiated dependency, and together these dependencies capture a large number of constraints and relationships frequently encountered in the business environment. The jurisdictional diagram specifies the authority hierarchy of agents. The hierarchies are based on roles and are used to allocate authorities and delegate policies.

**Social System / Dynamics**

The Conversation and Contract models describe the system’s behavior. The Conversation model identifies what messages are exchanged in order to fulfill the dependencies and jurisdictional relationships. This diagram lists all the messages an agent may receive and the possible responses to each one of these messages. The messages are based on speech acts.

The Contract model specifies the obligations (commitments) and authorizations between agents about the services provided to each other. An organization of agents is based on these commitments, and is used to facilitate cooperation and conflict resolution. A commitment means that the agent is willing to give access to its resources or services.
Yu, Du Bois, Dubois, and Mylopoulos describe in [Yu et. al 1995] an analysis and design framework that includes two languages: I* and ALBERT. I* was developed at the University of Toronto and ALBERT at the University of Namur. The authors did not explicitly distinguish the analysis and design phases. They did not relate the conceptual model to the design model, and in many cases there is a use of design entities in the analysis phase (see the “Account Handler” in the example provided in [Yu et. al 1995]). Nevertheless, I* can be considered as an analysis language (referred to by the authors as an “understanding level” model), and ALBERT a design language (referred to by the authors as a “specification level” model). I* is used to “support the generation and evaluation of organizational alternatives”, which is really the aim of the conceptual model – to present the current organizational structure and behavior or the future ones (while considering alternatives). The ALBERT (Agent-oriented Language for Building and Eliciting Real-Time requirements) is used to “produce requirements specification document for system developers”. A limitation of this framework is the absence of an “automatic” transformation mechanism from the I* model to the ALBERT model. Another limitation is that no guidelines are provided on how to map organizational reality to the models and answer questions such as “what entities should be represented by agents?”

This framework is requirements oriented, although it allows going back and forth between the conceptual and design models.

I* is a framework for modelling intentional relationships among strategic actors, and it is made up of two models: Strategic Dependency Model and Strategic Rationale Model. Both models deal with the dynamic nature of agent relations.

I* is represented in graphical and formal (using the Telos conceptual modeling language) forms. ALBERT supports the modelling of functional requirements by representing them

---

1 A more updated version of ALBERT (ABERT II) is described in [Du Bois 1995, Schbbens and Petit 1999]. We used the earlier version of ALBERT in this paper because it is integrated with I* for analysis and design.
as a society of agents, which interact in-order to fulfill an organizational need. ALBERT offers both a graphical representation and a textual one, which is based on logical formulas.

**Individual Agent / Static Structure**

In I* (the conceptual model), the internal structure of each actor is modeled using the following constructs: Beliefs, Wants, Abilities, Goals, Tasks, Commitments, and Resource. There is no description in [Yu et. al 1995] of the relations between these constructs.

In ALBERT (the design language) agents have states, which are a list of attributes and the values assigned to them (each attribute is specified as a table). States are used to describe the agent knowledge (of the external world and states of other agents). Actions performed by agents in order to discharge contractual obligations, and change their state. Agents are not intentional and do not have goals.

**Individual Agent / Dynamics**

In I*, the Strategic Rationale Model is used to describe the agent’s behavior. This model relates tasks, goals, and resources in order to allow the agent to formulate a plan, and lists two types of relationships:
- Means-end relationship: what other means exist for achieving the same task
- Task decomposition: how a task can be decomposed to several sub-tasks

In ALBERT (the design language) local constraints, which are defined as logical statements, define the admissible behavior of the agents:
- Effects on actions: the state changes that are performed by each action
- Causalities among actions: how the current state and an external action can trigger an agent’s action
- Capability – same as above, only that the action is not triggered; it is made possible
- State behavior: how state changes occur without actions (e.g., through time)
No mechanism is provided for allowing the agent to optimize his plans or for the agent to learn.

**Social System / Static Structure**

During the analysis phase the roles and positions of the agents are listed (in I*); However, there is no description of how the role and position hierarchies affect the agent’s behavior.

ALBERT does not include a description of the systems’ structure.

**Social System / Dynamics**

The second model in I*, the Strategic Dependency Model, represents interactions between agents (or strategic actors): who depends on whom to perform what actions; it presents an external view of how agents depend on each other. The model includes four types of dependency links:

- Task dependency: empowering other agents to perform a portion of a task
- Resource dependency: an agent (to perform a task) depends on the presence of another agent. The other agent is viewed as a resource.
- Goal dependency: other agents bring conditions to enable reaching a goal
- Softgoal dependency: same as goal dependency, but conditions are not defined strictly. The agent should consider the how conditions are met and decide accordingly.

The strength (status) of each dependency can be open, committed, or critical.

In the design process (in ALBERT) interactions take two forms: (a) making information visible to other agents and (b) one agent’s actions affect other agents. Constraints, too, have two types: local (described earlier) and cooperation constraint, limiting inter-agent interactions. The cooperation constraints are:

- Action perception: how and under what conditions other agent’s actions trigger own action
- State perception: the ability to see (know) other agent’s states
- Action information: what information about actions is made visible to other agents
- State information: what information about own states is made visible to other agents

The description of the system’s dynamics in I* and ALBERT is rich, but it does not include complex social interaction and notions such as negotiation and trust.
Agent-Oriented Methodology for Enterprise Modelling; Kendall et al.

Kendall et al. describe in [Kendall et. al 1996a] steps toward constructing a methodology for the analysis (enterprise modelling) and design of agent oriented information systems. The methodology is based on the integration of existing techniques, mainly the CIMOSA modelling framework for enterprise integration, the IDEF approach for workflow modelling, and the use case driven approach to object-oriented software engineering. Kendall et al. did a good job at comparing these approaches, integrating them, and adopting them to fit agent-oriented systems. The authors study the differences between objects and agents, and updates existing methodologies to accommodate these differences.

The general framework for the design of the agents’ cognitive map is based on the BDI approach, and agents are designed to reason and optimize their behaviors. There is no explicit notion of time in this framework. This agent-oriented methodology provides a graphical representation of the system with no formal semantics. Although this framework is said to support both the analysis and the design phases, there is no clear distinction between those phases and there are no well-defined guidelines on the A&D process and the use of all the diagrams.

Individual Agent / Static Structure

An agent perceives the environment via sensors. The perceptions update the agent’s knowledge about the world (his beliefs). An agent has goals, which represent states an agent wants to achieve. Based on these goals and his beliefs, he formulates a plan of actions. Plans are instantiated when an event occurs and conditions are met, and then it becomes an intention.

Individual Agent / Dynamics

The agent is able to reason and choose a plan, which turns into an intention. The agent’s intentions include three types of tasks: (a) internal task that updates the agent’s beliefs, (b) coordination task (interaction with other agents), and (c) invoking the effector, which impacts other objects. Beliefs change when a goal is achieved.
Sequence diagrams represent the conditions under which plans become intentions.

Use case diagrams are used to represent the workflow processes. IDEF diagrams specify the reasoning mechanism and the consideration used by the agent to select a plan.

**Social System / Static Structure**

Use case representation is a high level diagram, which connects all the use cases and the agents. This diagram shows what use cases could be integrated into which agent’s plan, and what use cases demand the cooperation of one or more agents. Also provided is a hierarchy of use cases that represents the hierarchy of work processes and a use case inheritance diagram.

There is no description of other structures and relation in the system, and generally speaking, the agent’s behavior is not constrained by the system’s structure.

**Social System / Dynamics**

The agent uses his sensors to interact with various objects via messaging. The agents’ interactions are modelled using diagrams, which represent the message workflow. However there is no description of the type of messages allowed (speech acts or others) and the message’s structure. Coordination protocols diagrams are used to specify the various reaction options an agent has in each conversation.

It is worth mentioning that the Kendall et al. extended this framework to create an architecture for designing multi-agent systems in [Kendall et. al 1996b]. The Layered Agent Pattern Language deals with reasoning agents, issues of cooperation and collaboration between agents, and mobility and translation between semantics (extremely important in open systems). The architecture is made up of seven different layers (or models), which interact with each other and altogether form a comprehensive framework for designing agent systems. However, this language is not an analysis and design methodology, and therefore not included in this paper.
Agent-Oriented Analysis and Design (AOAD); Birgit Burmeister

AOAD [Burmeister 1996] is an analysis and design methodology that is a natural extension to existing Object-Oriented techniques. AOAD provides models to support the analysis phase (where entities in the domain are identified and modelled as agents) and the design phase (where the behavior of agents is specified), but doesn’t make a clear distinction between the two phases. AOAD is made up of three models: Agent Model, Organizational Model, and Cooperation Model – all these models are used both to represent the organization (i.e., it’s a conceptual model, which is used during the analysis phase) and to describe the design of the information system (during the design phase).

The Agent Model is used to identify agents and their internal structure; the Organizational model is used to describe the static structure of the system – the class hierarchies of agents and agents roles, and the Cooperation Model represents the “dynamics in large” – the interactions among agents.

**Individual Agent / Static Structure**

The agent’s internal structure is described using the Agent Model. The model describes:

- The agents and the environment (which entities are agents and which are a part of the environment). Although the model is basically a conceptual model, it is also used to describe design-level entities (e.g., using agents to realize the system’s internal processes). There are no clear guidelines on how to determine which entity should be represented as an agent.

- The agent’s motivations – the interests, preferences, responsibilities, and goals of the agents, which drive their behavior.

- The agent’s knowledge and beliefs – knowledge about the external world and about other agents. This knowledge is used to execute the agent’s plans (see below)
The AOAD approach is not committed to a specific cognitive architecture, and the designer is free to choose the attributes to describe the agents.

**Individual Agent / Dynamics**

The “dynamics in small” are also a part of the Agent Model. This part of the model defines the agent’s behavior via plans. Plans are series of actions and are activated either to fulfill a motivation or as a response to an external event. Plans could be described graphically using state transition diagrams.

**Social System / Static Structure**

The system’s structure is defined by the Organizational Model, which includes the following steps:

- Identifying roles and responsibilities. Roles are mapped to agents, and influence the agent’s motivation and thus its behavior.
- Building inheritance hierarchy. Roles are classified into classes according to some attributes and agents with similar characteristics are grouped into one of these classes. This is very similar to OO techniques where classes are organized into a hierarchical structure.
- Structure roles into organizations by decomposing the system into sub-systems.

As with the previous model, this model too could be used in both the analysis and design phases, which makes the development process more ambiguous and less clear.

**Social System / Dynamics**

The system’s dynamics are represented in the Cooperation Model using the following steps:

- Identifying cooperation and partners. Cooperation between agents could be around a goal fulfillment, by sharing resources, or by synchronizing actions. The cooperation type, the reason of cooperation, and the cooperating agents are captured in this model.
- Identifying message types. The possible KQML message types to be used in the cooperation are captured in this model.

- Define cooperation protocols. Based on the two previous steps, the possible flows of messages among cooperating agents are captured in the cooperation protocols. Examples of such protocols are Informing, Querying, or Proposing.
MASB – Moulin B. and Brassard M.

MASB (Multi-Agent Scenario-Based) [Moulin and Brassard 1996] is an Analysis and Design methodology, which is specifically designed to support the development of multi-agent systems. MASB makes a clear distinction between the analysis and design phases, and provides distinct models and graphical diagrams for each phase. MASB borrows from theater and describes the reality using a metaphor of agents as characters playing roles in pre-defined scenarios. Complex behavior of human and software agents is captured by scenarios (scripts) – a way of specifying the agent’s social interactions, its behavior, and the knowledge used to specify the behavior. Agents have access to databases containing information that represents the world.

In the Analysis phase agent’s behavior is captured using scenarios. This phase is composed of five steps:

1. Scenario description (using natural language)
2. Role function description (using behavior diagrams)
3. Conceptual data modelling (using Conceptual Data and Entity/Object Life Cycle diagrams)
4. Static and dynamic description of the world
5. System-user interaction modelling

During the Design phase, the scenarios are refined and formally specified as agent’s behavior and knowledge structure that are used to implement the multi-agent system. This phase is composed of the following steps:

1. MAS architecture and scenario characterization
2. Object modelling
3. Agent modelling
4. Conversation modelling
5. Overall system design validation

The first step of the analysis phase describes the organizational environment as scenarios, in a textual form. This description should emphasize roles played by agents, the

...
information exchange, events that occur in the course of the scenarios, and actions taken by agents. An agent can play one or more roles, in one or more scenarios. Playing a role, an agent performs one or more activities.

The second step in the Analysis phase, role function description, specifies the agent-roles relations, as well as roles-activities, activities-information, and agent-environment relations. Behavior diagrams are used to represent the scenarios as a tripartite graph whose nodes are processes (activities), accumulations (information), and environment (the world and other agents). Edges in the diagram are flows (process-environment link) and channels (processes-accumulations link).

These two phases capture the behavior of the whole system (therefore cannot be classified into sub-categories) and are the starting point and the root of the analysis and design phases that follow. During the design phase the scenarios described in the analysis are used to generate design specifications, although in many cases these scenarios are further refined (e.g., by merging certain roles and splitting others, introducing new agents, and introducing detailed behaviors not considered previously). In the following, the analysis and design steps are detailed in the context of the representational sub-categories.

The notion of time is captured in MASB by Temporal Marks, which represent specific points in time or time intervals that may be significant for the agents. The agent has a clock and temporal marks are used to trigger actions.

**Individual Agent / Static Structure**

The agent’s knowledge is described in the behavior diagrams as accumulations. During the third analysis phase, data conceptual modelling, the accumulations are further detailed in terms of their attributes and the “data conceptual structure” is defined (using a graphical representation). This model describes the main elements that will be included in the agent’s database, and they could be represented using entity-relationship diagrams (ERD), object class diagrams, or “frames”. At any given time, the agent’s state is defined by the values associated with the attributes in the database. Another diagram, the entity (or object) life cycle diagram, is used to specify the allowed state transitions. An agent can deduce new knowledge (beliefs) from existing knowledge, based on a set of
deduction rules (which are contained in a special segment of the agent’s DB called the reasoning space).

In MASB, agents have the capabilities of intentional agents – they are able to record things as beliefs, make decisions by choosing goals to pursue, reason on their intentions and knowledge, and create plans of actions (aimed at achieving goals) and execute them. They, however, do not include an explicit model of other agent’s beliefs, intentions, and plans (as social agents do).

The mental state of an agent is defined by its beliefs, goals, and expectations. Expectations are the agent’s projection of future events, other agents’ actions, or its own behavior. Goals lead the agent’s actions and are organized in a hierarchy of sub-goals. During the design phase the agent’s belief structures are defined, based on the conceptual data modelling (the first step of the design). The agent’s knowledge is specified as belief structures or as objects shared by other agents and manipulated by the object server (see below).

**Individual Agent / Dynamics**

The agent’s actions (processes) are associated with the roles the agent plays. Actions are specified by different types of transition rules, which have preconditions, post-conditions, and co-conditions. These conditions are composed of attributes of mental states and message structures. Actions are organized into actions plans, which are activated in order to achieve a goal.

The decision space is the decision structure composed of goal hierarchies and relations between goals and beliefs. The decision space is used in the process of decision making and it includes different types of transition rules: activation rules (how a goal can activate its sub-goals), execution rules (conditions for execution of a goal), propagation rules (how the success or failure of a goal is propagated to other goals), and other rules that specify how a goal could be abandoned, suspended, or resumed.

In the third step of the design phase, the decision spaces for each role are specified and detailed (using a graphical representation). The actions space, which is used to describe the plans used by the agents, is also specified (and represented graphically).
Social System / Static Structure
Agents are associated with roles, which are specified in the behavior diagrams (second step of the analysis). Roles are associated with goals, plans, and actions. MASB does not include additional hierarchies (other than role hierarchy) to describe the structure of the system.

The data structures that characterise the world are described in the fourth phase of the analysis, static and dynamic descriptions of the world, and represented using conceptual data models (the same models used to describe the agent’s knowledge). Object servers contain that knowledge, which is shared and manipulated by agents. The world is modelled as a set of objects (reactive agents) that have simple reaction plans and are capable of evolving. In order to receive information about the world, agents send requests to objects contained in the object server.

Other types of agents (aside from the intentional software agents and object servers) are the scenario manager (which manages all the scenarios and directs agents actions at runtime) and the conversation interpreter (which enables the user-agent interactions).

During the design (second step), the objects structures are further specified, and the decision and action spaces of the conversation interpreter are detailed (fourth phase of the design – conversation modelling).

Social System / Dynamics

Agents communicate by exchanging messages (certain messages correspond to speech acts), and messages are also used to model external events that happen in the world.

The fifth step of the analysis phase describes the interaction between users and software agents.

MASB does not deal with complex social behavior (e.g., agents with contradictory goals) and its consequences (e.g., the need for a conflict resolution).
MAS-CommonKADS – Iglesias C., Garijo M., Gonzalez J. and Velasco J.

MAS-CommonKADS [Iglesias et.al 1996] is an analysis and design methodology for the development of multi-agent systems. It is, like CoMoMAS [Glaser 1996], an extension of the Knowledge Engineering methodology CommonKADS (which was developed by the European Community’s ESPIRIT program). Knowledge Engineering (KE) methodologies are similar in many ways to A&D methodologies: they provide models for representing the (organizational) domain and for designing the software system. Techniques developed in KE could be easily used in the analysis and design of information system, especially during the analysis phase, where knowledge about the static and dynamic nature of the environment is captured in a conceptual model. MAS-CommonKADS, thus, builds on the techniques and models of CommonKADS, and extends them by adding aspects that are relevant to multi-agent systems.

MAS-CommonKADS includes the following models to support the analysis phase:
- Organization model – the structure of the organization
- Task model – the general tasks and process that are performed in the organization
- Agent model – the capabilities and characteristics of the agents
- Communications model – user-machine interfaces
- Coordination model – interactions between agents
- Expertise model – the problem-solving knowledge used by an agent to perform a task

All these models, except the coordination model (and some modifications to the Agent model), are borrowed from CommonKADS.

MAS-CommonKADS includes one step prior to the construction of the analysis models – conceptualization. During this phase early requirements are described using use cases (based on object-oriented techniques). Also included in the methodology are guidelines for determining what entities in the environment should be represented as agents.

During the design phase, the Design model, which consists the following models, is constructed:
- Application design – decomposition of the system into modules and choosing an agent architecture for each agent (deliberative, reactive, or hybrid architecture). New types of agents may be introduced.
- Architecture design – the exact types and numbers of the agents are determined and a specific multi-agent architecture is selected (it includes agent types, agents’ knowledge and the use of ontologies, and coordination protocols to specify agents communication).

- Platform design – decisions on software and hardware to run the application

**Individual Agent / Static Structure and Dynamics**

The agent’s structure and dynamics are represented in the Agent model (analysis phase). This model represents services (facilities offered to other agents to help them satisfy their goals), goals (the objectives of the agents, which are satisfied by the reasoning mechanism), reasoning capabilities (different models could be used to represent reasoning), general capabilities (agents’ skills and the communication language they understand), and constraints (norms, preferences, and permissions).

The expertise model is used to represent the agent’s knowledge in three sub-levels: domain knowledge, inference level (inference structures), and task level (ordering of the inferences).

**Social System / Static Structure**

The Organization model is used to describe the system’s structure.

**Social System / Dynamics**

The Task model is used to represent general processes and tasks at the system level (before they are distributed to agents). The Coordination model is used to represent the agent interactions. These interactions are based on speech acts. The model contains of four elements: conversation (a set of interactions to ask for a service or request information from other agents), interaction (simple exchange of messages), capabilities (skills and knowledge of the participants in the conversation), and protocol (a set of rules that govern the conversation). Several graphical representation are used to represent the Coordination model: message sequence charts (to represent scenarios identified using the use cases; an alternative representation is Event Trace Diagrams), Event Flow Diagrams
(to model the generic behavior of the agent and the knowledge exchange in interactions),
and CEFSMs – Communicating Extended Finite State Machines (to model the control
flow in interactions, including message events, external events and internal events).
CoMoMAS – Glaser N.

CoMoMAS [Glaser 1996] is an environment to support the development of conceptual descriptions of multi-agent systems, and thus could be used as an A&D methodology. CoMoMAS includes tools and models to support the analysis (knowledge acquisition, modelling, and representation) and design phases, as well as a mechanism for automatically generating code, based on the design model. CoMoMAS, like MAS-CommonKADS [Iglesias et.al 1996], is an extension of the Knowledge Engineering methodology CommonKADS (which was developed by the European Community’s ESPRIT program). Knowledge Engineering (KE) methodologies are similar in many ways to A&D methodologies: they provide models for representing the (organizational) domain and for designing the software system. Techniques developed in KE could be easily used in the analysis and design of information system, especially during the analysis phase, where knowledge about the static and dynamic nature of the environment is captured in a conceptual model. CoMoMAS, thus, builds on the techniques and models of CommonKADS, and extends them by adding aspects that are relevant to multi-agent systems and by providing an automatic code generator.

CoMoMAS uses the commercial product KADSTOOL for the knowledge acquisition and modelling. The result of these activities is a set of conceptual models that need to be operationalized. Formalization of the analysis phase is obtained using the Conceptual Modelling Language (CML).

CoMoMAS is composed of four different modules to cover the whole knowledge engineering life-cycle: Acquire, Modeller, Constructor, and Coder. In this overview we will concentrate on the analysis and design activities realized by the first three modules. The Acquire is used for knowledge acquisition, where knowledge is represented in the form of transcription protocols (graphical and textual) and a glossary. In the Modeller, based on the protocols and the glossary, the CoMoMAS library is constructed. The library includes three types of knowledge categories: domain (concepts, properties, relations, and expressions), inference, and task, which are represented using: domain hierarchies, semantic networks, inference structure, and task hierarchies. Knowledge kernels (which include domain hierarchies, inference structures, and task decomposition,
and are partially translated into CML syntax) are then exported to the Constructor. The Constructor develops the design model and outputs the Coder model sets. The Coder then generates programming code, based on the model sets.

CommonMAS includes all the models of CommonKADS (as described in the MAS-CommonKADS section): Agent Model, System (Organization) Model, Task Model, Expert (Expertise) Model, Cooperation (Communications), and Design Model.

**Individual Agent / Static Structure and Dynamics**
The agent’s structure and dynamics are represented in the Agent model (analysis phase). This model represents: agent’s type, the agent’s architecture, roles, and communication protocols used by the agent. The Expert model specifies the problem solving techniques, cooperation methods, strategy, and behaviors of the agent.

**Social System / Static Structure**
The System (Organization) model is used to describe the system’s structure.

**Social System / Dynamics**
The Task model is used to represent general processes and tasks at the system level (before they are distributed to agents) and task hierarchies. The Cooperation model is used to represent the agent interactions, and it contains descriptions of conflict resolution mechanisms, cooperation primitives, and cooperation protocols.
Integrated Software Engineering (ISE) – Pont M. and Moreale E.

Integrated Software Engineering (ISE) [Pont and Moreale 1996] is an integrated methodology for process-oriented, data-oriented, object-oriented, and agent-oriented software development. The agent-oriented part is seen as the most general, thus we will describe only this portion of ISE. In general ISE is a high-level methodology, i.e., it describes the analysis and design process and provides some tools representing them, but does not contain explicit detailed procedures and models for describing the system.

ISE builds on well-established object-oriented techniques, integrates them and extends them to support the analysis (logical aspects – what the software is going to do) and design (a physical model) of multi-agent systems. There is a well-defined distinction within ISE between the analysis and design phases. The analysis phase contains four steps:

1. The external agents – identification of agents in the system’s environment
2. The logical interactions diagrams – definitions of agents’ conversations
3. The system and external messages – the system’s response to external messages
4. The message domain diagram (MDD) – interaction between the agents

The design phase includes the following steps:

1. The internal agent diagram – goals and services of internal agents are defined
2. The class relationship diagram (CRD) – relationships between agents
3. The physical interaction diagrams – description of processes

Basically, ISE is focused on the system level aspects, and the methodology is lacking tools for the representation of the agent’s cognitive model and behavior in the analysis phase. The description that follows is the analysis and design steps.

Individual Agent / Static Structure and Dynamics
The agents’ internal structure is not part of the analysis process (only for internal agents); it is only defined during the design phase. The first design phase, the internal agent diagrams, lists the agent’s aims (goals) and the services they provide. Other than that the agent’s structure and behavior is completely open-ended, and there is no cognitive architecture to guide the analysis and design of these aspects.

**Social System / Static Structure**

The external agents, their types and their goals are identified in the first step of the analysis process – the external agents. The hierarchy of the agent classes is represented by the class-relationship diagrams, as part of the second step of the analysis process – the logical interaction diagrams.

In the second design phase, the class-relationship diagram, the CRDs (borrowed from object-oriented methodologies) are constructed to represent the relationships between agents types.

The modelling of additional social structure (e.g., those based on roles or authority hierarchy) are not supported by ISE.

**Social System / Dynamics**

In the second analysis phase, the interaction diagrams are constructed. These diagrams represent the system’s behavior, including the control and timing of processes. Interactions between agents are modelled as conversations. The logical interaction diagrams represent the type of message passed, the initiator and the respondent, and the temporal order of the messages.

In the third analysis phase, the system and external messages, key external messages to which the system must respond, are identified in the logical interaction diagrams.

Message domain diagrams (MDD) are produced in the final analysis phase. These diagrams represent all the agents in the system and their interactions.

In the final design process, the physical interaction diagrams are defined based on the logical interaction diagrams. These diagrams define a series of methods (member function calls) – either in an implementation-independent fashion or in the form of code fragments.
Complex social interactions, which involve cooperation and collaboration between agents, are not considered in ISE.
AUML – the Agent UML; Odell J., Parunak H. and Bauer B.

AUML – the Agent UML (Unified Modeling Language)\(^2\) is an analysis and design methodology, which extends UML to represent agents [Odell et. al 2000]. AUML builds heavily on the foundations of UML, both in models representing the organizational environment (conceptual models) and in models used to design the information system (which we called implementation models). Agents are seen as extension of active objects, exhibiting *dynamic autonomy* (proactive action capability) and *deterministic autonomy* (the autonomy to refuse an external request).

The aim of AUML is to provide both a formal and an intuitive semantics, through a user-friendly graphical notation, for the development of agent-oriented systems.

AUML extends the UML mostly in one aspect (or one representation sub-category) – it provides models to capture the dynamic aspect of inter-agent interactions. The methodology does not provide extensions to capture the cognitive map of the agent (individual / static structure), or the structural organization of the agents (system / static structure).

AUML could serve to aid both the analysis and design processes, since it builds on existing models within UML, although the extensions proposed are mainly useful for the design process. The AUML extensions are made in the implementation model family (basically these are design models).

AUML provides extensions to the UML by adding a three-layer representation for agent interactions protocols (AIP), which define communication protocol as ‘an allowed sequence of messages between agents, and the constraints on the contents of these messages’ [Odell et. al 2000]. The authors chose to show how AIP could be used to adapt the UML to the agent paradigm, because agent-interaction are a complicated enough subset of the representation model, and thus could show the way for the entire adaptation of UML to support the development of agent-oriented information systems.

\(^2\) Please distinguish AUML from UAML [Treur 1999a, Luck et. al 1997, Treur 1999c]
The AIP levels are:

1. Templates and packages to represent the protocol as a whole.
2. Sequence, collaboration, and activity diagrams to capture inter-agent dynamics.
3. The internal agent processing, modelled by activity diagrams and state charts.

**Individual / Static Structure**
AUML does not provide extensions to UML to represent the internal cognitive map of the agent: no additional representation models are provided to capture the agent’s knowledge of the world and other components of its cognitive map, such as desires, intentions, and mobility mechanisms. Nevertheless, within UML there are several models that could be used to capture these aspects, mainly the OML syntax.

**Individual / Dynamics**
The 3rd level of the AIP captures the internal agent processing, and thus provides a mechanism to represent the dynamics of the agent’s cognitive map. AUML does not build on a specific cognitive architecture, and represents the agent’s internal dynamics using activity diagrams and state charts.

**System / Static Structure**
AUML builds on the class hierarchy included in the UML model, and add the notion of an agent playing a specific role. No other mechanism is provided to capture the system’s structure.

**System / Dynamics**
The first two levels of the AIP relate to the system’s dynamics, and could be used both in the conceptual and in the design models. In level 1, templates and packages provide reusable solutions for message sequencing (these are part of UML). The authors advocate the use of a set of packages as an extension to UML, which will capture the specifics of agent interactions. The AIP packages serve as templates, which are parameterized model elements. Level 2 contains both sequence and collaboration diagrams, which represent
the same semantics only in different graphical notation. Activity diagrams and state charts could also be included in level 2. Activity diagrams are used to represent processes, as they provide an explicit representation of the thread of control (which is useful in protocols that involve concurrent processing). State charts could be used to represent constraints for the protocol. Again, AUML includes extensions to UML’s diagrams (for sequence, collaboration, and activity diagrams) in order to model agent-based interaction protocols. The extensions include the use of communication acts and the support of concurrent threads of interactions.

AIPs could be specified in a detailed level, using a combination of diagrams (mainly activity diagrams), and thus capture the details of the interaction process. The more detailed the design models are, the easier it is to implement the systems.
Cassiopeia; Collinot A., Drogoul A. and Benhamou P.

Cassiopeia [Collinot et. al 1996] is a methodology for supporting the design of multi-agent systems. It focuses on system level aspects (mainly on the collective behavior of agents), and does not attend designing concerns related to the agent’s internal cognitive map.

Cassiopeia focuses on behavior as a central thread, and the design process in Cassiopeia has three steps, related to three levels of behavior:

1. Elementary – in this step the required elementary behaviors are defined, and based on them the agent types are defined.

2. Relational – the structural description of the agents organizations is defined (this is where relationships between behaviors are defined).

3. Organizational – the dynamics of agent interaction is described (organizational behavior).

According to the Cassiopeia approach, functional dependencies are inherent to the collective achievement of a task. The whole set of these dependencies determine the coupling of the organizational problem. Two types of couplings are possible: (a) static – when there is no competition between agents and the designer can define in advance the structure, and (b) dynamic- when there is competition and the designer only defines possible organizational structures, which are instantiated within the problem solving context.

**Individual / Static Structure and Dynamics**

Cassiopeia does not provide a model to support the representation of either the cognitive components of the agent, nor the dynamics internal to the agent.

**System / Static Structure**

Steps one and two of the design process describe the system’s structure. In step one, the agents types are defined, based on a given analysis model. In step two, the coupling graph is produced, representing the dependencies between agents. These dependencies are
termed influences. Then the relational behaviors that enable agents to identify and handle the influences are defined.

**System / Dynamics**

In design step 3, the behaviors that will enable the agents to manage the formation, durability, and dissolution of groups is defined. The occurrence of redundant groups indicates a redundancy of means to meet the need of a particular agent, named the trigger agent. As a way to control the formation of groups, the designer identifies the trigger agents (based on the influence graph) and determines selection methods for each. The designer defines the commitment signs that are produced by the trigger agents to indicate to other agents that a group is formed to meet their needs. The specification of these commitment signs defines the second group behavior – joining behaviors. The last group behavior to be defined is the dissolution behaviors. Cassiopeia does not provide a specific mechanism for describing the agent communications and interactions.
MESSAGE/UML; Caire et al.

MESSAGE (Methodology for Engineering Systems of Software AGEnts) [Caire et al. 2001] is an agent-oriented SE methodology, aimed to support the analysis and design of multi-agent systems. The MESSAGE project is funded by Eurescom, a research organization owned by European telecommunication companies. MESSAGE borrows from AUML [Odell et al. 2000] to include support for agents’ interactions, from other agent-oriented approaches (namely Gaia and MAS-CommonKads), and from goal analysis techniques [Mylopoulos 2001].

MESSAGE includes the following diagram types: organization, goal, task, deligation, workflow, interaction, and domain. The MESSAGE entities concepts are: ConcreteEntity, Activity, and MentalStateEntity.

The main Concrete entities are: agents (has services and purpose), organization (a group of agents working together towards a common purpose), role (a particular external characteristics of an agent in a particular context), and resource (non-autonomous entity used by agents).

The main activity types are: task (a task has pairs of situations describing pre and post conditions), Interaction (borrows from the Gaia methodology; has purpose and participants), and goal (associates an agent with a desirable situation).

The analysis process in MESSAGE starts from a top level decomposition, where the agent system is viewed as a set of organizations that interact with resources, actors, and other organizations. At this level the focus is on identification of entities and their relationships. From the top level the analyst proceeds to more detailed level of analysis, specifying the structure and the behaviors of the entities identified before.

**Individual Agent / Static structure**

Basically the internal structure of the agent is open-ended, and could be based on one of several cognitive architectures. MESSAGE separates the agent’s knowledge base from the inference mechanisms, but doesn’t specify the exact type of knowledge in the knowledge base.

However, MESSAGE does prescribe several constructs related to the agent’s internal structure, which are described in the Agent/Role view. An agent has a role (one or more),
and may use resources. For each agent or role, the view describes what goals it is responsible for, what events it needs to sense, what resources it controls, and what tasks it knows how to perform. The agent’s motivation is captured by the purpose attribute. The agent has goals that are either derived from its purpose or are intrinsic to the agent’s identity.

**Individual Agent / Dynamics**

An agent has services that describe his functional capabilities. Agent’s activities are tasks and interactions. An agent performs tasks, which are knowledge-level units of activity with pre and post conditions. Agent’s behavior is specified by a set of behavior rules.

**Social System / Static Structure**

The system’s structure is defined in the Organizational View, which describes agents, organizations, roles and resources, in one of several relationship: aggregation, power relationships (superior-subordinate relations), and acquaintance relationships.

The Goal/Task view describes goals, tasks, situations, and dependencies between them. They could be linked through logical dependencies to form graphs (i.e. describing goals decomposition and how tasks can be performed to achieve goals. Tasks decomposition is supported through causally linked sub-tasks.

**Social System / Dynamics**

Interactions model system behavior and agent coordination, and are described in the Interaction View. Each interaction has more than one participant (initiator, collaborators and the motivator) and a purpose the participants are collectively trying to attain. An interaction protocol defines a pattern of message exchanges associated with an interaction. A message is an object communicated between agents, and it includes the following attributes: sender, receiver, speech act (specifying the sender’s intent), and content.
Agent-Object Relationship (AOR); Wagner G.

Agent-Object Relationship (AOR) [Wagner 1999, Wagner 2000] is a design methodology for agent-systems. It also provides a description on how to transform these models into a database schema. The AOR architecture is inspired by Shoham’s AOP [Shoham 1993], and is an extension to OO methodologies.

Entities in AOR could be one of: object, agent, event, action, claim, or commitment. Agent classes are associated with event, action, claim, or commitment. An organization is viewed as a complex institutional agent, defining the rights and duties of its subagents.

Individual Agent / Static structure

AOR includes two basic types of entities: passive (objects) and active (agents). Similar to AOP, in AOR agents perceive the environment, and the cognitive state of agents is described in terms of beliefs, capabilities, choices, and commitments.

Individual Agent / Dynamics

Agents can communicate, act, make commitments and satisfy choices. Agents send and receive messages, and communication in AOR take the form of speech acts, according to a standard agent communication language.

Social System / Static Structure

There are three basic types of agents: artificial agents, human agents and institutional agents. Similar to OO modeling, agent and objects are organized in class hierarchies: specialization and component. An organization is an agent, composed of several subagents. The subagents have roles and positions. Duties to fulfill and rights to perform actions on behalf of the organization are associated with agent roles.

Social System / Dynamics

Agent’s interaction is described in terms of message exchange. Agent can take actions, are committed towards and have claims against other agents.
Open Distributed Application Construction (ODAC); Gervais and Muscutariu

ODAC is an Architecture Description Language (ADL) for defining the design of agent systems, and it aims at filling the gap between the analysis and design phases. It supports both the analysis, design, and implementation phases. ODAC design is compliant with the OMG MASIF mobile agent platform, thus it is not implementation independent.

ODAC is based on the ISO Open Distributed Processing (ODP) standard that define an architectural framework for the construction of distributed systems. ODAC is based on OO design methods, specifically on UML.

ODAC adopts the ODP reference model and defines five viewpoints to direct the development process: for the analysis phase: Enterprise, Information, and Computational views, for the design phase: Engineering view, and for the implementation phase: Technology view.

Individual Agent / Static structure & Dynamics

Based on UML design; No specific constructs are described.

Social System / Static Structure & Dynamics

In the Enterprise, Information and Computational views the behavior of the system is specified, as part of the analysis process. The objective of the system is detailed, the information that it handles and the tasks it carries out.

For the design model, the basic system level concepts are: role, community, objective, behavior and action. Agents are grouped into communities, and each community has policies governing its behavior. An agent is associated with a role, and behavior is associated with roles. Agents exhibit behavior that is required to realize the objective of their community.
The methodology presented by Bergenti & Poggi [2000] is an agent-oriented design methodology, and is implementation independent. It is an extension to UML, which utilizes UML’s customization capabilities to tailor the methodology to the AOIS domain. Agents are the atomic entities, and they interact with other agents through an agent communication language. Four agent diagrams are introduced:

- **Agent system architecture**: describing agent classes and their relations. Each class is characterized by the actions that an agent belonging to it can be requested to perform. Relations between classes may be used to express acquaintance relations.
- **Role diagrams**: specify the agent’s role in interactions, i.e. specifies the role that an agent plays in the protocol (see below)
- **Protocol protocols**: models agents’ communications over a set of roles.
- **Ontology followed by agents**: allow defining a model of the world composed of entities and relations. These relations are employed in agents’ communication to model the content messages that agents are allowed to use in communication.

These extensions to UML are focused on agents’ communication in an open and heterogeneous environment.

**Individual / Static Structure**

Two elementary constructs are used to describe agent (class): responsibilities and messages. B&P does not specify any additional constructs for modeling the agent’s cognitive map.

**Individual / Dynamics**

In general, this aspect is open ended and no specific constructs are pre-specified. For each agent, a set of actions is defined. These actions are mostly intended to support agents’ communication.
**System / Static Structure**
The system’s structure is defined by the set of agent classes and their relations. A specific relation is the acquaintance network, which is used to define which other agent each agent knows. Roles are associated with agent’s classes, and are utilized in agent interactions.

**System / Dynamics**
B&P extends UML by providing several diagrams for modelling agent’s communications. The agent’s communication capabilities depend on the role he plays. Interaction protocols model the flow of the interaction and the message contents, and the ontology defines the semantic of each message.
Agent-Oriented Role Based Workflow Modeling (AORBWM); Yu & Schmid

AORBWM is methodology for analysis and design of agent systems, which borrows from business process modeling. It makes clear distinction between the analysis and design phases, and provides guidelines to support the analysis and design processes. In AORBWM a business process is viewed as a collection of autonomous problem solving agents, which interact with each other and have inter-dependencies. Agent classes are associated with roles, which are defined in terms of goals, qualifications, obligations, permissions and communication protocols. Coordination of workflow is achieved through communication between agents.

AORBWM focuses on agents’ system structure, and provides comprehensive models for capturing the complexities of agents’ social structure.

Individual / Static Structure
Agents conduct conversations, perform activities, use resources and have capabilities. No constructs are specified for modeling the agent’s cognitive states.
At the design level, there are three types of agents: Personal Agents, Actor Agents and Internal Agents.

Individual / Dynamics
The actions the agent performs and the messages he exchanges are governed by the role he plays.

System / Static Structure
Agent classes are associated with roles, which defines a prototypical function for an agent in a workflow. Roles are defined in terms of goals, qualifications (necessary pre-conditions for achieving the goals), relationships (with other roles in the workflow), obligations (the role’s functionality: what activities the role must or must not perform), permissions (what activities the role is permitted to perform), protocols (the role plays a part in a set of protocols to achieve its goals), and resource (passively utilized during activity performance). Relationships among roles are categorized on two dimensions: authority (control relationships between two roles) and cooperation (the other roles to
cooperate with). One or more roles may be assigned to an agent. Agents are selected to play a role based on organizational policies and their capabilities.

Goals and activities could be decomposed, and are structured in a hierarchy.

**System / Dynamics**

Interactions between roles fall into typical patterns of message exchange, which are called protocols. A protocol is specified by a set of rules governing the conversation among agents. Conversation is a set of interactions for requesting a service or information. An interaction is a simple interchange of messages, and carries the following attributes: *speech act, communication language, knowledge representation language, synchronization, sender, receiver* and *ingredients*. 
Appendix B - Relevant Work in Neighbouring Fields

In Chapter 5 we referred the reader to fields of research where we might search for models or theories that could be employed as foundations for modelling multi-agent systems, specifically to Cognitive Psychology, Anthropology and Organizational Science.

However, there are other research areas where one might look for ideas about modelling agent systems. Several closely related fields that are interested mainly in designing systems, will not provide full theories to support agent modelling, but might provide some useful insights into the constructs and relations required to model agents. Researchers interested in developing agent-oriented A&D methodologies should look beyond the boundaries of this specific field, to neighbouring fields that address similar problems. Below we review work in closely related research fields: (1) agent-oriented programming languages and control architectures, (2) agent-oriented simulation and automatic prototype generation frameworks, (3) DAI agent-oriented problem solving frameworks, and (4) Computational Organizational Theory (COT). We will shortly describe each neighbouring field, discuss its similarities to work in agent-oriented analysis and design, and point out how work in that field might be relevant to our discussion. When describing framework in neighbouring fields, we will concentrate only on the representation models used in these frameworks, and will ignore other features.

Agent-Oriented Programming Languages and Control Architectures

In [Abbott 1987], programs are described as a representation of the domain knowledge (plus control), and programming languages as modelling schemes to represent that knowledge. Hence implicit or explicit model schemes in agent-oriented programming languages may serve as a basis for the design models in agent-oriented A&D methodologies.

For example, the model of the agent’s cognitive map within the programming language AOP [Shoham 1993] contributes directly to our understanding of how to design agent systems (and more specifically, how to design the agent’s internal structure). The mental state of the agent in AOP is defined by the agent’s beliefs (about the world, himself, and other agent’s mental state), capabilities (the ability to perform an action), and
commitments (or obligations). Although AOP is a programming language, it is easy to see how ideas about modelling and representation of agents used in this framework are relevant to the design of agent-oriented information systems. An example of an agent-oriented control architecture is Concurrent METATEM [Fisher 1996, Fisher 1999a, Fisher 1999b].

Agent-Oriented Simulation and Automatic Prototype Generation Frameworks
Techniques and tools under this category are not designed for solving problems, nor for creating a general framework for software engineering; they are designed for fast generation of multi-agent systems. Usually these multi-agent systems are used to simulate animal and human behaviours, to produce simple prototypes to test designs and architectures of agents, or even to develop full applications. Models within these frameworks focus mainly on design and implementation issues (and not on analysis issues), and hence provide representation that could be used to construct design models in A&D methodologies. Examples of frameworks in this field include DESIRE [Brazier et al. 1997a, Brazier et al. 1997b, Research Group AI, Vrije University, Jonker and Treur 1998], ASE – Agent Simulation Environment [Luck et al. 1997], STEAM [Tambe 1997], and SWARM [Minar et al. 1996].

DAI Agent-Oriented Problem Solving Frameworks
DAI (Distributed Artificial Intelligence) problem-solving frameworks 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. Models within these frameworks focus mainly on design and implementation issues and not on analysis. Some of these design models could be employed during the analysis and design process. Examples of frameworks under this category are TAEMS [Decker 1995, Prasaad at. al 1996], SIGAL [Maamar et al. 1997], and the “Language/Action Perspective” [Verharen and Weigard 1994, Verharen et al. 1997].
Computational Organization Theory (COT) Frameworks

COT is a field of science that uses the power of computers to study and simulate human behaviour\(^1\). COT frameworks are greatly influenced by theories and models from the social sciences. These methodologies borrow models from psychology and the behavioral sciences to model the agents’ internal processes, and ideas from organizational science to design the society of agents. Although COT frameworks are not generally thought of as analysis and design methodologies, they could easily be used to analyze any type of agent-oriented information systems. These frameworks rely heavily on the agent paradigm and provide models to support the analysis process. The methodologies that belong to this group are very expressive and include complex and comprehensive models of the agent’s cognitive behaviour and the behaviour of the agent society. Models of COT could easily be applied to the analysis phase (even if these models could not be applied as-is, they provide insight into what constructs and relations are suitable for representing organizational processes). These models comply with many of the guidelines sketched in this paper: they are founded on psychological\(^2\) and social theories, requirements driven, and comprehensive. Examples of COT frameworks are SDML [Moss et al. 1998, Carley and Gasser 1999 p. 321] and TOVE [Gruninger and Fox 1994, Fox et al. 1998].

\(^1\) A good introduction to COT can be found in [Carley and Gasser 1999] and [Epstein and Axtell 1996].

\(^2\) 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 field, as well as description of specific architectures.