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
ofarazy@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 [Elammarl 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].

In this paper, we describe different methodologies that are suitable for analyzing and designing AOIS. We view the analysis and design process as a modelling problem, and look at the representation models when studying the frameworks. In addition to agent-oriented Analysis and Design (A&D) methodologies, we also review the representational component of agent-oriented architectures, languages, and development frameworks. These neighboring fields provide valuable insight of what should be captured by models of an agent-oriented A&D methodology.

---

\(^1\) Please do not quote or cite without the authors’ permission.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 supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.
To better understand the various approaches, we analyze their strengths and weaknesses, classify them according to two dimensions. This then enables us to compile a list of requirements for an ideal unified agent-oriented modelling methodology.


General Terms: Design

Additional Key Words and Phrases: Autonomous agents, cooperative agents, multi-agent systems, agent-oriented information systems

1. Introduction

1.1 Background and Motivation

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

In order to apply the agent paradigm in large and complex industrial applications, and across various domains, tools for creating such systems are needed. One of the basic and most important tools in software development is an analysis and design (A&D) methodology. An A&D methodology supports the developer in the early stages of the software engineering (SE) process - the analysis and design phases.
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, Elammari and Lalonde 1999, Luck et al. 1997, Kendall et al. 1996a]. This task is one of the top priorities in the effort of turning agent-oriented software engineering into the prevailing software engineering (SE) paradigm, and agent-oriented IS into the prevailing paradigm for information systems.

1.2 Scope and Terminology

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). 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 DBMS technology. Systems such as those for supporting decision-making (e.g., expert systems and simulation systems) are within the scope, since they are also supporting 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.

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. 1996a, 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 form a multi-agent society;

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

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

Other characteristics of agents may only appear in some of the multi-agent systems, and thus are not part of the weak agency definition. For example, rationality, having mentalistic notions (e.g., desires, intentions, goals, and knowledge), 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.

In this paper, we review agent-oriented analysis and design methodologies as well as a few other agent-oriented frameworks, such as agent-oriented programming languages, agent-oriented development frameworks, and agent-based social simulations. This is because these frameworks include a representational component or a model of either the organizational environment or the software system, and these models could, in many times, be adapted to be used as either

---

3 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.

4 Only agent-oriented frameworks are included in the survey. 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 al. 1995, Arnold P. et al. 1991, Monarchi et al. 1992].
conceptual models\textsuperscript{5} (in the analysis phase) or design models (in the design phase). Other than the models used in these frameworks, we did not review additional important features of these frameworks, as they are irrelevant to the analysis and design processes. In section 2 we will give a few examples on how representational models within agent-oriented frameworks (which are not analysis and design methodologies) could be used in the system analysis and system design phases.

\textbf{1.3 Why This Survey}

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 which are not necessarily analysis and/or design methodologies (such as agent-oriented programming languages, implementation architectures, or agent-oriented simulation environments).

2. Over the past five years, many frameworks and techniques for developing agent-systems have been proposed. This paper identifies twenty 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,

\textsuperscript{5} The term ‘conceptual model’ has been used in the literature in different contexts. We adopt the definition of Wand and Weber \cite{Wand, Weber1995}, which relates conceptual model as a model of the real-world environment (and not a model of the information system). More details on conceptual models are given in section 2.
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 overcome these concerns and provide an up-to-date analysis of the field from a modelling perspective. 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.

Based on the analysis and the comparisons, we later compile a list of requirements for and sketch an ideal unified agent-oriented modelling methodology. We hope that this effort will be useful to researchers in developing new agent-oriented analysis and design methodologies.

This paper is different from the surveys done by Yu [1997] and by Iglesias, Garijo and & Gonzalez [1999]. Yu described only methodologies that support the analysis (requirements engineering) phase, and not the design or implementation phases. Iglesias et al. did not focus on analysis and design methodologies. Their survey included mainly implementation architectures and frameworks that are aimed at assisting the developers at the implementation phase. In addition, the dimensions that are used in this paper are different and the analysis is more detailed.

1.4 Outline of the Paper

The paper proceeds as follows: chapter two describes what agent oriented analysis and design methodologies are, chapter three lists the work included in the survey and assigns them to different classes, chapter four presents the analysis dimensions and maps the methodologies accordingly, chapter five sketches some guidelines to an ideal Unified Agent-oriented Modelling Language – UAML, and finally, our concluding remarks are given in chapter six. Appendix A

6 The term UAML is borrowed from [Treur 1999a]

7 Please distinguish UAML from AUML, a methodology reviewed later in this paper [Odell at. Al 2000]
includes a description of each one of the methodologies mapped earlier. The appendix is an integral part of the paper because it provides the reasoning and explanation behind the mappings.

2. Agent-Oriented Analysis and Design Methodologies

2.1 Methodologies and Models Provided By Them

An analysis and design methodology supports the processes of eliciting the system requirements, analyzing the organizational environment, and designing the information system. The early requirements and analysis phases deal with specifying the requirements of the system, while the design phase deals with the detailed specification of how the system will accomplish the requirements. In the analysis phase the application domain is modeled using abstract notions, and the conceptual model is generated. The conceptual model is a model of the organization. In the design phase the information system itself is modeled using concrete concepts, which relate directly to components of the software system. The design model should provide detailed information of 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].

Although various analysis and design methodologies use various names to describe the models and the phases, there is generally an agreement on the structure of the A&D process. It is seen as a collection of transformations, from the very abstract model (the conceptual model) to the most concrete model (the detailed design model), where each transformation shrinks the space of possible end products, and introduces more and more implementation bias. Ideally, the transformations should be straightforward, i.e., 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 has 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 list of
constructs and the 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 design model, on the other hand, includes constructs and relations that describe the structure and behavior of the information system. Both models should describe the system as a whole, as well as the cognitive map of the individual agent and the way each agent perceives the world.

The task of constructing a model is a difficult one, and consists of defining constructs and relations that are complete (capture every aspect of the system which is relevant for developing an IS) and non-redundant (constructs should not overlap). Moreover, tractability is also an important issue in determining what constructs to provide in a model. There exists a tradeoff between the descriptive power of the model and the ability to implement it in a computer (tractability) to solve problems.

Different methodologies use different models, which are based on different constructs. Examples of constructs are using Commitment, Roles and Permissions to represent the organization, and using Belief, Intention and Goal to describe the agent’s view of the world. Some methodologies are better at describing one aspect of the system, while others are better at describing other aspects.

All these raised two challenging questions. What models are suitable for analyzing and designing AOIS? What constructs and relations should the conceptual and design models include? To answer these questions, we cannot restrict ourselves from looking into only work done in the analysis and design methodologies. In the next few paragraphs, we give examples of how neighboring fields that provide agent-oriented representational constructs are relevant to A&D.

2.2 Relevant Models in Neighboring Fields

Although not an analysis and design methodology, models of organization used in the field of Computational Organization Theory (COT) capture the complexities of a business environment and could be used as conceptual models during the analysis of information system (even if these
models could not be applied as-is, they provide insight into what constructs and relations are suitable for representing organizational processes). For example, SDML [Moss et al. 1998, Carley and Gasser 1999 p. 321] is designed to model the dynamic aspect of organizational changes and to simulate the informal development of business processes, by building on time levels. The distinctive contribution of time levels to organizational modelling is in making it straightforward to represent organizational changes (both in structure and behavior). Pairs of low-level processes combine together to increase the value of their joint activity (relative to the value of their existing value). These structures are dynamic because the basic level of productivity associated with each process changes each period as a result of the enhancements generated internally within the business process. The combination of activities (business processes) is modeled as communication between agents. The communication is based on message exchange. Agents (activities) seek to combine with other agents in order to increase value. They are engaged in message passing which is aimed at finding the best activity to combine with. Activity Agents have endorsement scheme (formulated as rules), which is used to classify other agents as Reliable (or Unreliable) and Successful Collaborator (or Unsuccessful Collaborator) based on those other agents' past behavior. These ideas could, therefore, be very useful in understanding how to analyze agent-oriented information systems and how to develop conceptual models.

How are agent-oriented architectures used by DAI researchers to solve problems relevant to designing agent-oriented IS? The structure of TAEMS [Decker 1995, Prasaad at. al 1996] might answer that question. It has a model of an environment that generates episodes, which in turn comprise of objective task structures. TAEMS also includes a subjective model that describes how the objective structures map to the information available to the agents. This description of the environment is later used to design the coordination mechanism between agents. This modelling strategy is limited to one aspect of designing agent-oriented IS (it does not include an internal model of the agent), but nevertheless is might serve as the basis for a model of the complete system, at either the analysis or design levels.

COT and DAI problem solving architectures are not the only neighboring fields that one should look at for developing agent-oriented conceptual models (models of organizations) and design models (models of information systems) – other fields are Programming Languages and
Automatic Prototype Generation frameworks. Frameworks in both these fields include implicit models of the environment and of the agent’s cognitive map, and making them explicit could prove useful as a basis for developing design models. 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). Commitments could be either self-commitments (a decision or a choice to take an action) or commitments to other agents. Actions are determined by decisions or choices (which are obligations to oneself); Decisions are logically constrained by the agent’s beliefs; Beliefs refer to the state of the world (in the past, present, or future), to the mental state of other agents, and to the capabilities of this and other agents. Decisions are also constrained by prior decisions. Actions take place at various points in time, and depending on the time, have certain effects. The mechanism by which the mental state of the agent changes is the following: actions of agents are facts, agent updates his beliefs when perceiving a fact that contradicts his current belief. Obligations persist by default, but under some circumstances they are revoked (when the obligation is released by the other party or when the agent is incapable of executing his obligation). Decisions, which are obligations to oneself, persist by default, but can be revoked unilaterally, and capabilities are assumed to be fixed. Although AOP is a programming language it is easy to see how ideas about modelling and representation of an agent used in this framework are relevant to the design of agent-oriented information systems.

In all the neighboring fields described above there exist agent-oriented approaches, thus the models they provide may be relevant for the development of an agent-oriented analysis and design methodology. When studying framework that are not strictly A&D methodologies, we will concentrate only on the representation models used in these frameworks, and will ignore other features.

2.3 The Importance of Having an Analysis and Design Methodology

The importance of a structured A&D process, which can be found in most systems analysis and design books (e.g., [Kendall and Kendall 1999]), is twofold:
1. During development: a formal analysis and design process ensures that the system is developed according to its requirements. It makes implementation easier, and allows for fewer mistakes. The A&D process turns software engineering into a science and not an art. A&D methodologies provide the system analysts, designers, and programmers a common language to exchange ideas about the system and share common work processes.

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

2.4 Agent-Oriented vs. Object-Oriented

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

Due to the aforementioned differences between objects and agents, object-oriented methodologies cannot be used as is to develop agent-oriented information systems. While some ideas from object-oriented A&D techniques can be borrowed, there is clearly a need for new methodologies, which are rich enough to capture the structure and behavior of the individual

---

agent (e.g., cognition, intelligence, rational, and emotions) and those of the agent society (e.g., trust, fraud, commitment, and social norms).

3. What is Included in the Survey?

The methodologies, which will be analyzed in this paper, are listed in five tables: A&D methodologies, programming languages, automatic prototype generation frameworks, DAI agent-oriented problem-solving models, and computational organization theory (COT) frameworks (in alphabetical order).

3.1 Agent-Oriented Analysis and/or Design Methodologies

This group includes classical A&D methodologies (including workflow and business processes modelling techniques), which employ ideas from the system analysis and modelling fields. The methodologies, for example [Wooldridge, Jennings and Kinny 1999] and [Wand and Weber 1990], emphasize ideas such as the expressive power of the model, clarity and completeness, and stress the importance of a clear and unambiguous transformation from high level models (conceptual models) to lower level models (design models). Methodologies listed in the table below rely mainly on models and concepts from object-oriented analysis, design, and programming, as they currently are the most commonly used techniques, and the object-oriented framework is the closest one to agent orientation.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Methodology</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bradshaw et al</td>
<td>KaOS</td>
<td>Bradshaw et al. 1997</td>
</tr>
<tr>
<td>Brumeister</td>
<td>Agent-Oriented Analysis and Design (AOAD)</td>
<td>Burmeister 1996</td>
</tr>
<tr>
<td>Coltell and Chalmeta</td>
<td>CafeAGS</td>
<td>Coltell and Chalmeta 1999</td>
</tr>
<tr>
<td>Collinot, Drogoul and Benhamou</td>
<td>Cassiopeia</td>
<td>Collinot et al. 1996</td>
</tr>
<tr>
<td>DeLoach</td>
<td>MaSE</td>
<td>DeLoach 1999</td>
</tr>
<tr>
<td></td>
<td>Methodology</td>
<td>Reference</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Elammari &amp; Lalonde</td>
<td>High Level / Intermediate Models</td>
<td>Elammari and Lalonde 1999</td>
</tr>
<tr>
<td>Ferber and Gutknecht</td>
<td>AALADIN</td>
<td>Ferber and Gutknecht 1998</td>
</tr>
<tr>
<td>Gadomski</td>
<td>TOGA</td>
<td>Gadomski 1993</td>
</tr>
<tr>
<td>Glaser</td>
<td>CoMoMAS</td>
<td>Glaser 1996</td>
</tr>
<tr>
<td>Iglesias, Garijo,</td>
<td>MAS-CommonKADS</td>
<td>Iglesias et al. 1996</td>
</tr>
<tr>
<td>Gonzalez and Velasco</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kendall, Malkoun &amp;</td>
<td>AO Methodology for Enterprise Modelling</td>
<td>Kendall et al. 1996a</td>
</tr>
<tr>
<td>Jiang</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kinny, Georgeff &amp;</td>
<td>A Methodology and Modelling Technique for Systems of BDI</td>
<td>Kinny, Georgeff and Rao 1995</td>
</tr>
<tr>
<td>Rao</td>
<td>Agents (KGR)</td>
<td>Georgeff and Rao 1995</td>
</tr>
<tr>
<td>Odell, Parunak and</td>
<td>AUML</td>
<td>Odell et al. 2000</td>
</tr>
<tr>
<td>Bauer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moulin and Brassard</td>
<td>MASB</td>
<td>Moulin and Brassard 1996</td>
</tr>
<tr>
<td>Pont and Moreale</td>
<td>Integrated Software Engineering (ISE)</td>
<td>Pont and Moreale 1996</td>
</tr>
<tr>
<td>Verharen E. and</td>
<td>Language/Action Perspective</td>
<td>Verharen and Weigard 1994</td>
</tr>
<tr>
<td>Weigard H.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wagner G.</td>
<td>AOR (Agent-Object-Relationship)</td>
<td>Wagner 1999</td>
</tr>
<tr>
<td>Wooldridge, Jennings &amp; Kinny</td>
<td>Gaia</td>
<td>Wooldridge, Jennings and Kinny 1999, Wooldridge, Jennings and Kinny 2000</td>
</tr>
<tr>
<td>Yu, Du Bois, Dubois</td>
<td>I* and ALBERT</td>
<td>Yu et al. 1995</td>
</tr>
<tr>
<td>and Mylopoulos</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.2 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 models schemes in agent-oriented programming languages may serve as a basis for the design models in agent-oriented A&D methodologies. Below is one programming language (AOP) that uses the notion of an agent as the fundamental concept (similar to an object in the OO paradigm), and one agent control architecture (Concurrent METATEM), which could be used as a programming language to develop multi-agent systems.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Methodology</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoham</td>
<td>AOP</td>
<td>Shoham 1993</td>
</tr>
</tbody>
</table>

3.3 Agent-Oriented Simulation and Automatic Prototype Generation Frameworks

Techniques and tools under this category are not interested in solving problems, nor in creating a general framework for software engineering; they are interested in fast generation of multi-agent systems. Usually these multi-agent systems are used to simulate animal and human behaviors, 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.
### 3.4 DAI Agent-Oriented Problem Solving Frameworks

Architectures listed below 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.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Methodology</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decker &amp; Lesser</td>
<td>TAEMS</td>
<td>Decker 1995, Prasaad et al. 1996</td>
</tr>
</tbody>
</table>

### 3.5 Computational Organization Theory (COT) Frameworks

COT is a field of science that uses the power of computers to study and simulate human behavior. COT frameworks are greatly influenced by theories and models from the social sciences. These methodologies use models from psychology and the behavioral sciences to model the agents’ internal processes, and ideas from organizational science to design the society of agents. Although COT frameworks are not generally thought of as A&D 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. However, the design process is not supported because the final outcome is a simulation rather than an actual system being constructed. The methodologies that belong to this group are very expressive and include complex and comprehensive models of the agent’s cognitive behavior and the behavior of the agent society.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Methodology</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox et al.</td>
<td>TOVE</td>
<td>Gruninger and Fox 1994, Fox et al. 1998</td>
</tr>
<tr>
<td>Moss, Gaylard, Wallis and Edmonds</td>
<td>SDML</td>
<td>Moss et al. 1998</td>
</tr>
</tbody>
</table>

4. Mapping the Frameworks

As we have mentioned in the introduction, only agent-oriented techniques are included in this analysis. We tried our best to include all the available agent-oriented analysis and design methodologies as well as several other agent-oriented frameworks. A description of the analysis dimensions and a mapping of the methodologies are presented next.

4.1 The supported software engineering phase

We believe that the clearest and most important distinction among the various methodologies is the software engineering process that they support [Treur 1999a, Mylopoulos 1999]. Frameworks that are not strictly A&D methodologies are mapped according to the representational model they include and the software engineering phase it could be employed in.

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

---

9 Agent frameworks and implementation tools that are not included in this survey, are: MADE [O’Hare and Wooldridge 1992], ARCHON [Cockburn and Jennings 1996], AgentSpeak [Weerasooriya et al. 1994], JACK [Busetta et al. 1999], ADE [Kupries 1999], ZEUS [Ndumu et al. 1999], KARO [Meyer 1998, ISYS Group 1999], DECAF [Graham and Decker 1999], MadKit [Gutknecht and Ferber 1999], TuCSoN [Omicini and Zambonelli 1998], Lightweight Agent Framework [Kindburg et al. 1999], SIF [Funk et al. 1998], MASSYVE [Rabelo et al. 1999]. A more comprehensive overview of agent-oriented development tools could be found in [Reticular Systems 1999] (a survey of commercial products and academic research program), and in [Wickler 1999].
**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 look after implementing the system?”

**Design:** the design phase (also referred to as ‘formal specifications’ phase) specifies the architecture of the future information system that will be built according to the definitions in the conceptual model. The end result of this phase is a design model, which is “an accurate and formal statement that expresses the desired behavior of the information system [Coltell and 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.

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

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

All frameworks that are considered analysis and/or design methodologies are presented in Table 1 according to the SE phase that they support.
### Table 1 – The supported SE phase for A&D methodologies

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Analysis</th>
<th>Design</th>
<th>Implement.</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CafeAGS</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>AALADIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOGA</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>KGR</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>Language/Action Perspective</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>Agent-Oriented Role-Based Workflow Modeling (AORBWM)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation supporting techniques could be seen as a type of 4GL. Methodologies which support this phase are not really A&D methodologies; they are software architectures or implementation framework for fast generation of agent-oriented systems. We include them in the analysis because:

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

The rest of the frameworks (the ones that are not analysis and/or design methodologies) are presented in Table 2 according to the SE phase they support.

<table>
<thead>
<tr>
<th>Methodology</th>
<th>The supported SE phase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analysis</td>
</tr>
<tr>
<td>Concurrent METATEM</td>
<td></td>
</tr>
<tr>
<td>AOP</td>
<td></td>
</tr>
<tr>
<td>DESIRE</td>
<td></td>
</tr>
<tr>
<td>ASE</td>
<td></td>
</tr>
<tr>
<td>SIM_AGENT</td>
<td></td>
</tr>
<tr>
<td>STEAM</td>
<td></td>
</tr>
<tr>
<td>SWARM</td>
<td></td>
</tr>
<tr>
<td>TAEMS</td>
<td></td>
</tr>
<tr>
<td>TOVE</td>
<td></td>
</tr>
<tr>
<td>SDML</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 – The supported SE phase for other agent-oriented frameworks
Two comments about Table 2 are worth mentioning. First, most frameworks in that table are intended to support only the implementation phase, but they provide valuable models and representations that could easily be adapted to serve as design models. Second, although it is not intended as such, TOVE is mapped in the ‘analysis’ category because the models it provides for modelling organizations could be used as conceptual models during the analysis process.

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

4.2 Abstraction Granularity

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

Fine grain models are restricted frameworks in that they make use of specific constructs and specific relations among them, while coarse grain models may leave a few degrees of freedom to the system designer. The advantage of fine grain models is that they are clearer and easier to implement, while open-ended models are less restrictive and could be tailored to the specific needs of the designer. A good example of these two ends is the design of the internal structure of an agent. Methodologies based on the BDI\(^{10}\) model [Kinny, Georgeff and Rao 1996, Georgeff and Rao 1995] define the agent’s cognitive mechanism using only three attributes: Beliefs, Desires, and Intention (BDI), while other methodologies may allow the designer flexibility in choosing the attributes and the relations between them. Approaches such as BDI, which use well-known and tractable models\(^{11}\), are necessary when the cognitive model of the agent must provide an optimal action plan. Less restrictive models with greater expressive power are necessary when

---

\(^{10}\) 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.
there is a need for a more accurate cognitive model of the agent (similar to human), for instance in simulations of human behavior.

Since a methodology may include a fine grained model of one aspect of the system (for instance the agent’s internal structure) and an open ended model of another aspect (for example the social dynamics), it is useful to further divide the domain of multi-agent modelling into several sub-categories (aspects), and then to examine the granularity of the models in each of them.

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

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

The top left cell, *Individual Agent / Static structure*, relates to the way the cognitive (and emotional) map of the agent is structured: what constructs are used (e.g., beliefs, desires, and goals) and what are the relations between them. We will use the KaOS methodology [Bradshaw et al. 1997] to demonstrate this analysis. KaOS includes the following attributes of the agent’s cognitive map: Knowledge (Beliefs and Facts), Desires, Intentions, and Capabilities. Facts represent Beliefs in which the agent has confidence about; Facts and Beliefs may be held privately or be shared. Desires represent goals and preferences that motivate the agent to act, and

---

11 Tractable models are models that can be solved by computers in a finite time and using finite storage spaces.
Intentions represent a commitment to perform an action. However, emotions and constructs that relate to the emotional state of the agent are not included in this model.

The bottom left cell, Individual Agent / Dynamics, deals with the functionality of the agents, the way in which the cognitive map evolves (e.g., updates of beliefs), the reasoning mechanism the agent employs (for planning), and his learning capabilities. In KaOS, the agent’s dynamics model includes a description of how the cognitive map is updated as the agent goes through his life cycle: birth, life, and death (also a Cryogenic state). During their lives agents read, process, and send information continuously and according to the information they receive they may update their internal structure. However, there is no description of a planning or scheduling mechanism, and the agents seem to be acting directly on basis of their intentions, without considering the implications of their actions and other agents’ actions on the environment. Learning is also not part of this model.

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

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

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

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

The mappings are provided in Tables 3 and 4, and use the following legend:

- **Fine grain**;
- **Coarse grain**;
- **Open ended**.

Table 3 maps the analysis and/or design methodologies according to their granularity in each of the representational sub-categories, while Table 4 maps the rest of the frameworks. The classes we used in the tables are fine grain (detail models that allow little flexibility in defining new constructs), coarse grain (not so detail models that may allow some flexibility in defining new constructs), and open-ended (no model at all, but usually with lots of flexibility in defining constructs). We are aware that this classification is rather crude, but we believe that it provides a good starting point to look at different methodologies. Readers interested in a specific class can then refer to the appendix for a detailed analysis of those methodologies.
<table>
<thead>
<tr>
<th>Methodology</th>
<th>Individual level</th>
<th>System Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure</td>
<td>Dynamics</td>
</tr>
<tr>
<td>KaOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CafeAGS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cassiopeia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High Level / Intermediate Models</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AALADIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOGA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoMoMAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAS-CommonKADS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AO Method. for Enterprise Model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KGR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUML</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language/Action Perspective</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I* and ALBERT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent-Oriented Role-Based Workflow Modeling (AORBWM)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 – Representation granularity for A&D methodologies
Table 4 – Representation granularity for other agent-oriented frameworks

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

Although these concepts are important, they are not an inherent part of multi-agent systems and the degree to which they are needed depends greatly on the application domain. Thus, the detailed analysis of the methodologies in the appendix does not include these aspects or categories.
5. Sketching an Ideal Unified Agent-oriented Modelling Language (UAML)

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

“The field of agent-oriented systems is growing dramatically in many directions. Coupled with its relative youth, however, this has given rise to concentration of research in distinct niches so that there are very different approaches to essentially similar problem areas with, in some cases, little or no interaction. Such a fragmentation leads to a lack of consensus regarding such fundamental notions as agents and autonomous agents, and also impedes progress toward integrated approaches to agent theory and agents construction. As the field matures, the broader acceptance of agent-oriented systems will become increasingly tied to the availability and accessibility of well-founded techniques and methodologies for system development”

The UAML should include models of both the individual agent and the agent society and of both the static structure and the dynamics of the system. When developing the future A&D methodology, several important issues should be addressed:

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

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

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

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

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

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

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

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

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

We believe that by providing more structure and complex relations at the system level, we will create more constraints over the behavior of the agents, and thus make it easier for the agent to formulate a plan. We should not reduce the expressive power of the model just to make it tractable; the model should be as expressive as possible, and at a later stage (during implementation) techniques and heuristics for planning and scheduling in that environment should be developed. Generating an optimal plan should not be a concern at the preliminary phases, where the conceptual and design models are formulated. This promises an architecture that is independent of the agent’s reasoning model. In applications where rational agents are needed, a translation between the reasoning-independent model and the reasoning model should be developed. Again, the methodology provided by Wooldridge, Jennings, and Kinny [Wooldridge, Jennings and Kinny 1999] complies with this requirement.
We also believe that the methodology should not restrict the constructs that are used to model the agent’s cognitive map, nor should it restrict the relations between them. SIM_AGENT [Sloman and Logan 1999] is an example of such architecture.

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

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

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

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

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

The field of Computational Organizational Theory (COT) can provide much knowledge and insight for constructing the foundations of the UAML. As mentioned earlier, models of COT could easily be applied to the requirements and analysis phases. These models comply with many of the guidelines sketched above: they are founded on psychological\(^{12}\) and social theories, requirements driven, and comprehensive. A good introduction to COT can be found in [Carley and Gasser 1999] and [Epstein and Axtell 1996].

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

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

\(^{12}\) 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.
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.

Jan Treur states [Treur 1999a]:

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

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

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

This appendix includes a description of each one of the methodologies classified in section four13. Each description includes an overview of the methodology, its aim and the theoretical and logical foundation it employs. In addition the granularity of the methodology in each of the four representational sub-categories mentioned in chapter three is evaluated. The ways in which the methodologies cover each sub-category vary greatly. Some methodologies provide a comprehensive and detailed model of the cognitive structure of the agent (Individual Agent / Static structure), while others focus on the interactions among agents (Social System / Dynamics). We use the following legend to indicate how the methodologies relate to each cell:

- Open-ended - the methodology does not include a specific model to guide the analysis or design, and overlooks this sub-category. In the case of open-ended implementation frameworks, some may allow free coding in those places, for instance a framework may provide no model for implementing agents interaction, but allow the user to freely code any such interaction.

- Coarse grain – the methodology includes some model to support that specific sub-category, but the model is not detailed.

- Fine grain – the methodology includes a detailed model, which is completely committed to a specific (psychological) theoretic architecture.

For example a methodology that provides a comprehensive and detailed model of the agent’s internal structure and of his behavior, but provides only partial description of the system’s dynamics will be classified the following:

13 The following methodologies are NOT described in the appendix (yet): TOGA (Gadomski), AALADIN (Gutknecht and Ferber), Language/Action Perspective (Verharen and Weigard), AOR (Wagner), Cassiopeia (Collinot, Drogoul and Benhamou) and Rule-Based Workflow Modelling (Yu and Schmid)
Following is a description and an evaluation of the agent-oriented frameworks. We analyzed each methodology according to the information available in the papers referenced. 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. 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.
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. Emotions and constructs that relate to the emotional state of the agent are not included in this model.

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. Learning is also not part of this model.

**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.). 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.

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.

Another important aspect of agent interoperability in an open environment is bridging semantic gaps. Speech acts provide a mechanism of sharing a common syntax among various agents, but it does not solve the problem of semantic differences. For example when two agents, designed by different people, say “may be”, they can mean different things: one agent may mean “probably yes”, while the other may mean “probably no”. This problem is much more complex than it seems, and although no current solution is 100% suitable, some approaches that use ontologies are providing a reasonable solution.
Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Structure</td>
<td>Coarse grain</td>
<td>Open-ended</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Coarse grain</td>
<td>Fine grain</td>
</tr>
</tbody>
</table>
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, which 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 tools for 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. Also it does not provide a technique for describing the agent’s planning or scheduling capabilities.

**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.

Another issue, which is not dealt with in Gaia, is the problem of interactions between agents
from different systems, which migrate in an open system environment. To allow interoperability
in such an environment it is crucial that the agents have a way of sharing the same semantics.

Following is mapping of the granularity of the model in each of the four representational sub-
categories:
<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-ended</td>
<td></td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
TAEMS; Decker and Lesser

TAEMS is a framework and a simulation environment for dealing with task intensive AI tasks, which was developed by Decker and Lesser [Decker 1995, Prasaad at. al 1996]. The model is intended at optimizing the processing of tasks via distribution of the problem to a network of processors (agents). This is a DAI approach for solving problems, via an agent architecture. Performance is judged by execution time and by quality. The model is a formal one, and a simulation environment is built on it. The agents are very simple (processors): their internal structure includes beliefs (knowledge) and actions, and the social interactions are limited in nature.

“Using TAEMS framework involves building a generative model of an environment that produces episodes that in turn comprise objectives task structures. A subjective model describes a mapping these objective structures to the information available to the agents. All of this together is the description of the environment that can be used to design and analyze coordination mechanism.” [Decker 1995].

TAEMS in not an analysis and design methodology, nevertheless, some ideas used in the TAEMS framework might be useful in constructing a methodology for agent-oriented analysis and design of information systems.

**Individual Agent / Static structure**

The cognitive map of the TAEMS agent is a very simple one. An agent has a state; it has some knowledge (or belief) about the task structure, and based on this knowledge and the task he has to perform he decides to execute his own methods.

**Individual Agent / Dynamics**

The functionality of the agents is defined by the set of methods. An agent has a strategy, which is constantly being updated, of what methods he intends to execute at what time.
Social System / Static Structure

The TAEMS framework does not include a model for describing the static structure of the social systems, and it does not include constructs such as Roles, Skills or Resources.

Social System / Dynamics

The only social activity is that of decomposing and distributing the tasks. Agents are not engaged in negotiations or in any other form of complex social interactions, and there is no notion of interoperability.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Structure</td>
<td>Coarse grain</td>
<td>Open-ended</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
BDI based Agent-Oriented Methodology (KGR); Kinny, Georgeff and Rao

The KGR methodology [Kinny, Georgeff and Rao 1996, Georgeff and Rao 1995] is based on the BDI (Beliefs, Desires, Intentions) approach for designing agent-oriented systems. KGR 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).
The internal model does not include emotional states.

**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. There is no notion of Rights (privileges), Resources, or Skills.

**Social System / Dynamics**

The social activity is described in the Interaction Model, where the syntax and semantics of the interactions is defined.
There is no treatment of complex social interactions and mechanism needed for interoperability in an open environment.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fine grain</td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
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].

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.

There are no additional hierarchies that relate to the agent classes, such as roles or skills hierarchies.

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.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-ended</td>
<td>Coarse grain</td>
<td></td>
</tr>
<tr>
<td>Coarse grain</td>
<td>Coarse grain</td>
<td></td>
</tr>
</tbody>
</table>

High Level and Intermediate Models; Elammari and Lalonde

Elammari and Lalonde introduced their agent-oriented methodology in the Agent99 conference at Seattle [Elammari 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 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 Elammari 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.

Following is mapping of the granularity of the model in each of the four representational sub-categories:
<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coarse grain</td>
<td>Fine grain</td>
</tr>
</tbody>
</table>

| Dynamics         | Coarse grain     | Fine grain    |
Yu, Du Bois, Dubois, and Mylopoulos describe in [Yu et. al 1995] an analysis and design framework that includes two languages: I* and ALBERT\textsuperscript{14}. 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 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.

\textsuperscript{14} 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.
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.
Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Structure</td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Fine grain</td>
<td>Fine grain</td>
</tr>
</tbody>
</table>
The TOVE Project; Fox et al.

The TOVE project is an effort to model the enterprise in order to explore alternative process designs in business reengineering. This project could be classified as belonging to the research field of Computational Organization Theory (COT), where information systems are employed in order to model simulated organizations. The evaluation of the TOVE project is based on [Fox et. al 1998] by Fox, Barbiceanu, Gruninger, and Lin and also on [Gruninger and Fox 1994] by Fox and Gruninger.

Although TOVE is not an A&D methodology, the models and languages it uses could be easily employed during the analysis process. It does not provide tools to support the design process. The TOVE grammar is a modelling methodology that employs the Artificial Intelligence perspective for inferencing and deduction.

TOVE focuses on the system structure and behavior and not on the individual agent. Furthermore, the overall optimized scheduling plan of the system is reached through centralized mechanism and not as an emergent property of agents that optimize their local plans (in this respect, TOVE is very different from most multi-agents systems approaches).

The TOVE grammar is based on ontology or a list of constructs and the relations between them. The concepts included in the model are:

- **Organizational hierarchy**: Divisions and sub-divisions
- (organizational) **Goals**, sub-goals, their hierarchy (using AND & OR), and dependency of goals.
- **Roles**, their relations to skills, goals, **authority**, and agents. Connection to **processes** (or activity network) that have been defined to achieve the goal and **policies** (constraints over the performance of processes). Authority over usage of resources, activities, and status changes (activities and their states). Generalization and specification of roles (**classes and subclasses**)
- **Skills**, and their link to roles
• **Agents**, their affiliation with teams and divisions (and home division), link to roles and activities and their means of communication. **Commitment** to achieving goals assigned to the role the agent plays. Authority over other agents (through roles). **Empowerment** as the right to change status (of activities and their states).

• **Communication links** between agents: sending and receiving information. Communication with authority (over goals, resources, empowerment, and roles) to create **obligation**. Communication at three levels: information, **intentions** (ask, tell, deny…), and **conventions** (semantics). Levels 2 & 3 are designed using speech act.

• **Teams** as temporary group of agents

• **Activities** and their **states**, the connection to resources and the constraints. **Status** of activity and activity state. Activities are defined as states changing mechanism.

• **Resources** and their relation to activities and activities states

• **Constraints** on activities (what activities can occur at a specific situation and a specific time)

• **Time** and the duration of activities. Actions occur at a point in time and they have duration.

• **Situation** (the relations to other constructs is unclear)

The relations between the constructs are presented in the following figure (taken from [Fox et. al 1998] p. 139)
These relations describe the static structure of the system. The link between agents represents the dynamic behavior of the system.

There is an explicit notion of time in TOVE: activities have duration and they take place at a certain point in time. This allows for scheduling and planning of the future.

**Individual Agent / Static Structure**

TOVE does not explicitly define properties of the agent, but sub-states can be seen as properties that together define a state. An agent has a list of possible actions, he plays a role, and through the role, he has authority, goals, and skills. There is no notion of constructs such as knowledge, desires, intentions, or beliefs, and we do not think that the agent has autonomous decision-making and action capability; these are only defined by the system’s constraints.
Individual Agent / Dynamics

Changes in an agent’s state space – state transformations are being done through actions. There is no notion of external events and it is unclear whether and how actions of one agent can affect the state of another.

The relations between activities and states are depicted below (based on [Fox et. al 1998] p. 137):

Each action has an enabling state, a state that must occur for the action to take place, and a caused state, the state of the system after the activity is performed. A state can be divided into sub-states, each representing one variable. The assignment of values to all sub-states defines the parent state. A parent state is referred to as “non-terminal” and a sub-state is referred to as “terminal”.

Social System / Static Structure

This is the heart of the TOVE model and it provides many hierarchies and relations to capture the structure of the enterprise. All the relations in the first diagram, excluding the inter-agent communications links, define the system’s structure, and they include: goals and their hierarchy, organizational formal structure, Roles hierarchy and their relations to goals, roles and skills relations, roles and authority relations, agent and roles relations, agent and organizational
structure relations, agents team relations, agent activity relations, and activity resource relations. These relations are used to constrain the agents’ actions.

**Social System / Dynamics**

Social activity of agents and their communication is limited to the communication links (sending and receiving information). The communication includes authority (over goals, resources, empowerment, and roles) and is aimed to create an obligation. An agent can specify the information he is interested in as well as the information he is willing to expose to other agents. The communication has three levels: information (the contents of the message), intentions (ask, tell, deny…), and conventions (semantics). Levels 2 & 3 are designed using speech act.

The communication links in TOVE provide a comprehensive and rich mechanism for defining inter-agent interactions.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse grain</td>
<td>Fine grain</td>
<td></td>
</tr>
</tbody>
</table>

| Dynamics         | Fine grain       |
|------------------| Fine grain       |
Agent-Oriented Programming (AOP); Shoham

Shoham introduced a new programming language in 1990 and later in 1993 [Shoham 1993] – Agent Oriented Programming, which is based on Object-Oriented Programming. AOP includes data structures and data components that relate directly to a set of modalities (or constructs), which define the agent’s mental state. Shoham focuses on the description of the individual agent, his mental state, and how this state changes over time.

Although AOP is not an A&D methodology, this framework could contribute to our understanding of how A&D methodologies should be designed. Shoham describes the constructs that should be used to design the agent’s mental state, as well as the constructs that should be used for inter-agent communications (based on speech acts theory). Time is included explicitly in AOP and each belief or commitment of an agent take place at a certain point in time and relate to an event (future or past) at a certain point in time.

Shoham’s first version of AOP is called AGENT0 and the logical component of this language is a quantified multi-modal logic.

Below is an analysis of the representation model, which Shoham uses to capture the behavior of the agent (and not an analysis of AGENT0):

**Individual Agent / Static Structure**

The mental state of the agent 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). Commitments could be either self commitments (a decision or a choice to take an action) or commitments to other agents.

There is no notion of emotions in this model.

**Individual Agent / Dynamics**

The agent’s dynamics define how actions are performed and the way in which the mental model evolves. Actions are determined by decisions or choices (which are obligations to oneself);
Decisions are logically constrained by the agent's beliefs; Beliefs refer to the state of the world (in the past, present, or future), to the mental state of other agents, and to the capabilities of this and other agents. Decisions are also constrained by prior decisions. Actions take place at various points in time, and depending on the time, have certain effects.

The mechanism by which the mental state of the agent changes is the following: actions of agents are facts, agent updates his beliefs when perceiving a fact that contradicts his current belief. Obligations persist by default, but under some circumstances they are revoked (when the obligation is released by the other party or when the agent is incapable of executing his obligation). Decisions, which are obligations to oneself, persist by default, but can be revoked unilaterally. Capabilities are assumed to be fixed.

In [Shoham 1993], there is no description of how and if the agent is capable of reasoning and planning.

Social System / Static Structure

There is no representation of the system’s structure in this framework.

Social System / Dynamics

Agents interactions are communicative actions that are modeled using speech acts. Messages could be one of three: ‘request’ (to perform an action), ‘unrequest’ (to refrain from action), and ‘inform’ (to pass information).

This part of the model is not comprehensive and the model does not support the representation of complex social behaviors.

Following is mapping of the granularity of the model in each of the four representational sub-categories:
<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine grain</td>
<td>Open-ended</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamics</th>
<th>Fine grain</th>
<th>Coarse grain</th>
</tr>
</thead>
</table>
DESIRE; The Research Group Artificial Intelligence at Vrije University

DESIRE (framework for DEsign and Specification of Interacting REasoning components) [Brazier et. al 1997b, Research Group AI, Vrije University, Brazier et. al 1997a, Jonker and Treur 1998] is a research program at the Vrije University, which commenced in 1989.

DESIRE is a framework for designing multi-agent systems, supporting all development phases, starting from the requirements phase all the way to the implementation. It can be considered as a meta-model since the framework could be used to model various agent systems with different cognitive and interaction models. DESIRE could be used to develop scheduling and planning applications in complex domains or as a framework for simulating physical processes, animal behavior [Jonker and Treur 1998], or human social behavior [Brazier et. al 1997b]. The logical foundations of the framework are based on a temporal approach to the formal semantics of reasoning behavior.

The heart of DESIRE is the tasks and their attributes. There is a hierarchy of tasks and tasks are directly related to components (agents). Tasks have input and output knowledge and tasks are related to each other in temporal relations (expressed in temporal logic rules). Information links between components are used to specify information exchange between tasks, and the sequence of tasks is explicitly modelled within components as task control knowledge [Brazier et. al 1997a].

The framework includes several generic models, which are used to capture both the static and dynamic aspects of the individual agent and the agent society. The generic models are Own Process Control, Agent Specific Tasks, World Interaction Management, Agent Interaction Management, Maintenance of World Information, and Maintenance of Agent Information [Brazier et. al 1997a, Jonker and Treur 1998]. In [Brazier et. al 1997b], two additional models are introduced: Maintenance of History and Cooperation Management. There are generic information links between the various models, and these links define the agent’s cognitive process – from perception to action. All the above mentioned models are described below.
**Individual Agent / Static Structure**

The agent’s knowledge is maintained by three models: Maintenance of World Information (knowledge of the external world), Maintenance of Agent Information (information of the agent’s characteristics), and Maintenance of History (information about past observations and interactions). Other constructs that are necessary in order to design the agent’s cognitive model are defined in the Own Process Control model.

**Individual Agent / Dynamics**

The agents reasoning mechanism is defined in the Own Process Control model, where constructs such as belief, goal, and intention could be defined. Since DESIRE is a meta-model, the system is flexible of representing any reasoning mechanism (see [Brazier et. al 1997a] for an example of implementing the BDI model).

The performance of the agent’s specific tasks is defined in the Agent Specific Tasks model. Interaction with the external world (the way in which external events effect the cognitive map of the agent and the way by which the agent’s actions effect the external world) are defined in the World Interaction Management.

**Social System / Static Structure**

During the analysis phase, the domain ontology is constructed and the hierarchies of various concepts and the relations between them are identified and are expressed as predicate logic rules.

**Social System / Dynamics**

The Agent Interaction Management and the Cooperation Management models are used to define the social interactions. The Agent Interaction Management component is responsible for message exchange between agents and the Cooperation Management is responsible for managing cooperativeness.
The interactions between agents do not follow any specific language or interaction protocol (such as the protocols based on speech acts), and allow the developer to choose any forms of messaging.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse grain</td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
The SIM_AGENT Framework; Sloman et al.

Aaron Sloman and others researchers of the Cognition and Affect project at the University of Birmingham have constructed a toolkit for fast generation of multi-agent systems. This toolkit, SIM_AGENT [Sloman and Logan 1999], could be used to explore various designs and architectures of agents as well as for simulations of animal and human behavior. SIM_AGENT focuses on the design of the agents’ cognitive model and is unique in that it does not commit to a specific architecture (e.g., BDI or SOAR). The toolkit allows the developer to define the constructs that define the agent’s reasoning (and feeling) mechanism and the relations between those constructs.

Individual Agent / Static Structure and Dynamics

SIM_AGENT provides a very generic framework for defining the agent’s structure and behavior. Basically agents perceive the environment, process the perceptions, and perform actions that effect the environment. Several mechanisms are available for processing the perceptions and deciding on an action:

- Reactive processes: very simple transformations of perceptions to actions, which allow the agent to act fast.
- Deliberative processes: planning, deciding, and scheduling. When employing these processes, motives are explicitly generated and plans are created. Learning is also possible. These processes are much slower than the reactive processes.
- Alarm mechanism: allows for rapid redirection of the whole system. This mechanism is used when the environment changes rapidly and there is a need to trigger stereotyped internal and external responses.
- Meta management: represent reflective processes and allow the agent self monitoring, self evaluation, and self control of his internal processes

SIM_AGENT doesn’t embody any particular psychological theory and the user can design agents that employ only some of the processes or all of them. Agents that employ only reactive processes could be used to simulate body reflexes. Agents that employ reactive processes and an
alarm act as if they have “primary emotions” (see the works of Damasio and Picard [Picard 1997]) and are able to avoid sudden dangers and take advantage of sudden opportunities. Agents with “secondary emotions” employ deliberative processes in addition to reactive processes, and are able to express complex behavior such as feeling relieved when danger has passed or switching modes of thinking.

**Social System / Static Structure and Dynamics**

SIM_AGENT focuses on the individual agent and there is no representation of the system’s structure. Inter-agent interactions are unstructured and open ended, and the user is free to code any communication messages.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static Structure</strong></td>
<td>Coarse grain</td>
<td>Open-ended</td>
</tr>
<tr>
<td><strong>Dynamics</strong></td>
<td>Coarse grain</td>
<td>Open-ended</td>
</tr>
</tbody>
</table>
ASE - Agent Simulation Environment; Luck, Griffiths and d’Inverno

ASE [Luck et. al 1997] is an agent simulation environment that is used for fast prototype generation and for experimenting with agent design and architectures. The simulation is specified in the Z specification language and is implemented in C++ in an object-oriented fashion. This framework focuses on the internal view – the individual agent, and it doesn’t cover issues regarding the social system’s structure and behavior.

In ASE, agents are treated as specific objects. The framework includes four types of entities: basic entity, objects, agents, and autonomous agents. There is no explicit notion of time, and this is a limitation of Z.

**Individual Agent / Static Structure**

Each of the four types of entities is designed as a special case of the more basic class (e.g., agents are a specific case of objects, and hence inherit objects’ attributes). Entities have attributes, objects have attributes and actions, agents have goals, and autonomous agents have motivations.

There is no restriction of the types of attributes used to define the agent’s knowledge and his cognitive processes.

**Individual Agent / Dynamics**

Objects have no internal control and they respond directly to the environment; they can perform actions and change the environment. Agents have goals, which drive the intelligent behavior and lead to actions. ASE includes simple reactive agents that use a pre-defined sequence of actions to achieve a goal. The agents should be able to perceive the environment (or part of it) in order to trigger actions. In the ASE framework perception is defined as a function of the goals, and is defined individually for each agent. When more than one action is possible the agent chooses an action based on the action-selection function; this function should be defined by the developer. Autonomous agents have motivations, which are related to goals. Motivations have strength levels, and these could be designed to change in reaction to the changing environment. As the motivation passes a threshold, an appropriate behavior is performed.
The simulation runs for a user specified number of iterations, and on each iteration the agents perceive the environment (based on the perception functions) and form their own world view. Levels of motivation are updated (if necessary) and initial conditions for each behavior are checked. Active behaviors are performed and the environment changes. This scenario accounts for one iteration.

The agents’ cognitive map does not include a deliberative level (a mechanism for allowing proactive behavior; see [Research Group AI, Vrije University] for a discussion of the different cognitive levels), thus the agents are incapable of planning or scheduling.

**Social System / Static Structure**

The ASE framework does not include any description of the static structure of the system.

**Social System / Dynamics**

There is no description of inter-agent interactions or any messaging method, so the agents are assumed to interact only with the environment.

Following is mapping of the granularity of the model in each of the four representational subcategories:

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse grain</td>
<td></td>
<td>Open-ended</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Fine grain</td>
<td>Open-ended</td>
</tr>
</tbody>
</table>
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.
Following is mapping of the granularity of the model in each of the four representational sub-categories:

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

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.
CafeAGS; Coltell and Chalmeta

CafeAGS is an environment for prototyping and formal verification of real-time and multi-agent information systems based on the agent-oriented paradigm [Coltell and Chalmeta 1999]. The framework includes a language and an environment for describing the system specifications in formal statements (design model), an automatic generation of a prototype, and verification of the results against the requirements. The approach is based on object-oriented techniques and on formal theory of dynamic logic. This framework supports the design process and does not include a language and models to support the analysis process (description of the requirements).

The components of CafeAGS are AGSS (Agent System Specification) and AGAVS (Agent Automatic Verification System). AGSS includes AGRSL (Agent Requirements Specification Language), an environment for translating the AGRSL into a prototype, and AGSI, a dynamic logic interpreter that executes the prototype. The prototype generated by the translator is based on formal theory of dynamic logic.

AGRSL is composed of textual sentences, which describe the agent classes, agent attributes, actions, and constraints.

What follows is an analysis of AGRSL, the requirements specification language.

Individual Agent / Static Structure

After agents have been identified, the attributes and action of each agent are described. The internal structure of the agent is very similar to the way it is described in object-oriented approaches (attributes and actions), and there is direct means of specifying the agent’s beliefs (knowledge), goals, or intentions.

Individual Agent / Dynamics

The dynamics (and some of the static structure) of the agents is defined in the agent template. The agent template declaration has three parts: Signature, Constraints, and Perceptions. Signature declaration specifies the state components (attributes) and actions performed by the agents. Every action has time duration, and by default agent can view each other’s actions. Actions may
be composed of simpler actions, and the action hierarchy is expressed in Process Algebra form. Actions may also have priorities.

Constraints are temporal formulas that are used to restrict possible agent life. They include dynamic integrity constraints (to restrict sequence set), valuations (which allow knowing the final state after executing an action), cause-effect relations (set a chain of actions occurrence), duties (trigger an action under certain conditions), and prohibitions (restrict an action occurrence under certain conditions).

The perception part indicates when agents do not perceive other agents’ states and actions. By default, agents are able to perceive each other’s actions, however in certain cases, agents can hide their actions.

The structure and behavior of the CafeAGS agents is very similar to those of objects, and no mechanism is available in this framework to allow the agents to reason and plan.

**Social System / Static Structure**

Agents are organized in classes and communities. The agent community declaration is composed of the community’s definition, the global domain declaration, and the agent classes that belong to it. The agent class definition includes the class name, the agent template, and a list of all agents belonging to that class.

Other than these hierarchies, no other model is provided to describe structures and relations in the system. Constraints over the agents’ behavior are not derived from system’s structures and are defined directly in the agent template.

**Social System / Dynamics**

Agents’ interactions are based on actions (same as objects) and there is no mechanism for structuring the conversations (e.g., using speech acts) or for describing complex social interactions.
Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static Structure</strong></td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
<tr>
<td><strong>Dynamics</strong></td>
<td>Fine grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
STEAM; Tambe M.

STEAM [Tambe 1997] is a multi-agent architecture that is especially designed to support teamwork. The STEAM (a Shell for TEAMwork) architecture is based on Soar [Laird, Newell and Rosenbloom 1987], and was extended by providing constructs and operators to allow agents to coordinate their joint work. Carley and Gasser relate to this framework as a Multi-Agent Soar approach [Carley and Gasser 1999]. This framework is not an analysis or a design methodology; it is an implementation architecture.

The social theory behind this architecture is the joint intentions theory [Levensque, Cohen & Nunes 1990], and STEAM enables the explicit representation of team goals and plans, and team’s joint commitments. Agents in this framework are able to reason about their own, as well as about the team’s goals and plans.

STEAM’s basis is in executing hierarchical reactive plans, and in order to include joint plans, it introduces the notion of team operators. These operators include pre-conditions rules, applications rules, and termination rules.

Individual Agent / Static Structure

The internal structure of the agents is based on the Soar model [Laird, Newell and Rosenbloom 1987]. STEAM includes the following additional constructs to represent teamwork: joint intention and joint commitment (which is defined as a joint persistent goal - JPG).

STEAM maintains agents’ private state by applying private operators and team states by applying team operators.

Individual Agent / Dynamics

Again, the basic dynamics are based on Soar. The following extensions enable teamwork: a joint intention is based on a JPG; a JPG to perform a team task P requires that all team member agents mutually believe that P is false and want P to become true (P is a WAG – weak achievement goal). Team members cannot decommit P until it is mutually known to be achieved, unachievable, or irrelevant.
Social System / Static Structure

There is no description in [Tambe 1997] of any social structure that is used in this framework.

Social System / Dynamics

Team members must synchronize to achieve the JPG, and they do so via the request-confirm protocol (Smith & Cohen 1996). The key is a persistent WAG (PWAG), which commits a team member to a team task P prior to the JPG. In addition to synchronization, this protocol enforces some important constraints (such as having the team member re-transmit a message in the case the message did not get through).

STEAM integrates some decision theoretic communication selectivity that reduces the communication load.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamics</td>
<td>Fine grain</td>
<td>Fine grain</td>
</tr>
<tr>
<td></td>
<td>Fine grain</td>
<td>Fine grain</td>
</tr>
</tbody>
</table>
The SWARM simulation system [Minar et. al 1996, Burkhart R. 1997] was developed at the Santa-Fe institute for the study of complex system (it emerged from work in computational biology). This simulation environment, which could be classified as part of COT research [Carley and Gasser 1999, Epstein and Axtell 1996], uses computer simulation to study the behavior of complex systems. The SWARM beta version was released in 1995 and was one of the first simulation environments built around the agent paradigm. The main idea is that complex behavior can emerge from the interaction of simple entities. SWARM is not an A&D methodology; it is a simulation environment, which adopts the modelling formalism of agents interacting via discrete events. SWARM is a generic framework, with no domain specific requirements, agent representations, or interaction patterns.

The basic unit in SWARM is an agent. SWARM is not committed to a specific model of the agent’s behavior, but SWARM agents are usually simple reactive entities. Agents are grouped into SWARM, an entity that includes the agents as well as the scheduled activities of those agents. SWARMs can be structured in a hierarchy, and the entire environment is composed of SWARMs. A SWARM can itself be an agent and its behavior is defined by the emergent behavior of the group of agents.

Carley and Gasser state that one of the intended goals of SWARM is to run Artificial Life (A-Life) applications. SWARM could be used to grow realistic looking social behaviors, and show how simple entities, through interactions, can come to exhibit collective intelligence [Carley and Gasser 1999].

SWARM is implemented in Objective C, an object oriented language. An agent is modeled directly as an object, types of agents are objects classes, and the state of the agent is modeled as the instance variables of the object.

---

15 ALife is a field of study where mathematical models and computer systems are used to model and simulate real-life phenomena, such as biological systems. This field of science is closely related to COT. See [Epstein and Axtell 1996 pp. 17-19] for a discussion of ALife.
Individual Agent / Static Structure

SWARM offers a very generic and flexible way of designing the agent’s internal structure. Agents perceive the environment (stimuli) and act on the environment (responses). Rules relate the stimuli to actions. The agent’s beliefs (or knowledge of the world) is modeled as a SWARM. Agents may own a SWARM, and the entities and interactions in that SWARM represent the agent’s knowledge of the world.

Individual Agent / Dynamics

Agent’s actions are implemented as object-oriented methods. Individual actions take place at some specific time and time advances only by events scheduled at successive times. A schedule of discrete events on the agents defines a process occurring over time. A schedule is a data structure that combines actions in the specific order in which they should execute. So the agents are basically reactive entities and the planning and scheduling takes place at a higher level (the SWARM).

Social System / Static Structure

The only social structure provided by the simulation environment is the SWARMs hierarchy.

Social System / Dynamics

Since agents in SWARM are implemented as objects, the agent’s behavior is limited to activities of simple objects. There is no mechanism to support agent conversations, coordination, or interoperability.

Following is mapping of the granularity of the model in each of the four representational sub-categories:
<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Structure</td>
<td>Coarse grain</td>
<td>Open-ended</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Coarse grain</td>
<td>Open-ended</td>
</tr>
</tbody>
</table>
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. Overall, all these models are coarse grain, and they do not restrict the system analyst to a specific set of constructs.

**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.

Following is mapping of the granularity of the model in each of the four representational subcategories:

<table>
<thead>
<tr>
<th>Static Structure</th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
SDML – Moss S., Gaylard H., Wallis S. and Edmonds B.

SDML (Strictly Declarative Modelling Language) [Moss et. al 1998, Carley and Gasser 1999 p. 321] is a programming language optimized for modelling multi-agent systems within certain articulated social structures, such as organizations. SDML is situated within the research fields of Computational Organizational Theory (COT), where computational tools are used to study social phenomena. SDML, therefore, is a programming language, which is designed to facilitate flexible multi-agent modelling of organizations, more strictly speaking, SDML provides a tool for studying and analyzing organizational process. The language has object-oriented features and corresponds to a fragment of strongly grounded autoepistemic logic. Unlike other COT approaches, which build on specific cognitive architectures, such as Soar, SDML is theory neutral with respect to the agent’s cognitive map, and it could be used to represent sophisticated agents and the nature of their social relations.

SDML is designed to model the dynamic aspect of organizational changes, by building on time levels, whereby different databases are associated with agents at different periods in time. Time levels enable the user to specify appropriate frequencies of interaction (which may differ between agents). The distinctive contribution of time levels to organizational modelling is in making it straightforward to represent organizational changes (both in structure and behavior).

Limitations of SDML as a framework for the study of social phenomena are:

a. Interactions among business processes are modelled strictly as numerical values and the generic representation of output is “productivity”.

b. The organizational hierarchy is purely binary.

c. The organizational structure is based only on the organizational hierarchy. Other structures, such as friendship network and authority hierarchies, are not modelled.

d. Agents are not goal directed. The notion of bounded rationality, where agents have limited resources to complete their tasks, does not exist.

e. No mechanism is available for planning and for the agent to assess the future impact of their current actions.
From a Software Engineering perspective, this language could not be directly applied as an Analysis and Design methodology, although certain features and concepts in this framework could prove very useful. Different from other COT frameworks, which usually provide models for analyzing the organizational processes (Analysis phase) and a simulation environment to realize those models, SDML does not include separate models for representing the organizational environment (and the models are incorporated in the language). Thus SDML is not suitable as an Analysis methodology. However, SDML could be generalized to be used as an agent-oriented programming language for implementing agent systems for any application domain.

**Individual Agent / Static Structure**

Since SDML does not build on any cognitive architecture, its mechanism for describing the agent’s cognitive model is open, and is based on rules. Rules have antecedents and consequences, and both consist of clauses. Agents have a rulebase associated with them, which is used for making decisions.

The agent’s knowledge (and beliefs) is represented as facts in the database. The knowledge may or may not be shared with other agents.

The SDML model has a very limited representation of cognition – agents evaluating each other as possible collaborators (through message exchange).

**Individual Agent / Dynamics**

Agents communicate by writing the results of their rule firing to their databases and reading the results of another agent’s rule firing from that agent’s database (agents may or may not allow other agent’s to read their database). This is the exact mechanism by which the agent’s knowledge is updated. Agents cannot perceive the results of other agent’s actions in the same period.
Social System / Static Structure

The social structure is based on the simple hierarchical organizational structure, and implemented as a hierarchy of object classes. Inheritance is realized through two principles hierarchies: type hierarchy (the type Agent is distinguished from the type Object in that it has a rulebase associated with it) and the organizational hierarchy, which is implemented as a container hierarchy. An agent inherits rules from both its super-classes.

There are several predefined agents types and these types are implemented as a hierarchy of classes (as in OO). Composite Agent and its subtypes (Serial, Parallel, and Merging Composite Agents) facilitate the representation of multi-agent structures, Looping Agents iterate over time and are used by agents that have rulebases for different time-levels, Meta Agent have the ability to write rulebases and thus provide means for implementing learning.

The container hierarchy is used to represent the organizational hierarchy, from departments down to individual agents, where each department is a Parallel Composite Agent. At the lowest level of the organizational hierarchy are business processes, which are represented by simple agents (termed Activity Agents) - cognitive agents with rules for making decisions. This organizational hierarchy is formed by a binary tree, where pairs of low level business processes (modeled as simple agents) form a higher level processes (modeled as complex agents)

Thus there is no real distinction between the agent and the activities he performs.

Social System / Dynamics

SDML has been designed to simulate the informal development of business processes with the organization. Pairs of low-level processes combine together (as described above) and increase the value of their joint activity (relative to the value of their existing value). These structures are dynamic because the basic level of productivity associated with each process changes each period as a result of the enhancements generated internally within the business process.

The combination of activities (business processes) is modeled as communication between agents. The communication is based on message exchange Agents (activities) seek combine with other
agents in order to increase value. They are engaged in message passing which is aimed at finding
the best activity to combine with.

There is nothing in this architecture that prevent agents for not being truthful with respect to their
activity’s value, and each agent (activity) has to decide whether a potential collaborator is honest
and reliable. Activity Agents have endorsement scheme (formulated as rules) which is used to
classify other agents as Reliable (or Unreliable) and Successful Collaborator (or Unsuccessful
Collaborator) based on those other agents’ past behavior.

This allows the simulation of complex organizational environment, where phenomena such as
trust and fraud could be studied.

Following is mapping of the granularity of the model in each of the four representation
subcategories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Structure</td>
<td>Open ended</td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Open ended</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
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 run-time) 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)
Following is mapping of the granularity of the model in each of the four representational sub-categories

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Structure</td>
<td>Fine grain</td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Fine grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
Concurrent METATEM – Fisher M.

Concurrent METATEM [Fisher 1999a, Fisher 1996, Fisher 1999b] is a high level logical language to describe the architecture of agent systems. The language, which is based on temporal logic, could be used as a programming language for the development of multi-agent systems (it provides a logical representation of agent systems, that could be directly executed). Thus Concurrent METATEM cannot be considered an A&D methodology, but certain concepts and models that are used by the language could serve as a basis for an agent-oriented design model (in [Wooldridge and Muller 1999], it is classified under “Agent Control Architectures” and not under “Agent-Based Software Engineering”). The following short overview will concentrate on the concepts of Concurrent METATEM that are relevant from an A&D perspective.

Concurrent METATEM focuses on the agent’s internal structure, and provides a three layers architecture to represent the agent’s behavior: a reactive layer, a deliberative layer, and a cognitive/social layer.

The basic tenets of Concurrent METATEM are:

1. Everything is an agent
2. All agents are concurrently active
3. The basic mechanism for communication between agents is broadcasting

Time is represented explicitly using a discrete and linear model of time, thus time is modelled as an infinite sequence of discrete states. An agent’s behavior is represented by a temporal formula using temporal rules.

Individual Agent / Static Structure

There are no predefined attributes that define the agent’s cognitive model, thus the internal structure of the agent is open-ended. Agent’s behavior is specified by temporal rules (see below).
**Individual Agent / Dynamics**

The agent’s behavior is described using the different layers: the reactive layer includes immediate interactions with the environment, the deliberative layer includes medium-term symbolic planning, and the social/cognitive layer includes longer-term elements such as coordination and cooperation activities. Each level has different types of constraints and works on different time scales. The layers interact and thus produce the agent’s behavior.

The representation of the agent’s behavior is based on temporal logic (extended with knowledge belief operations and refined to handle various deliberations). (Temporal) rules define the behavior: reactive rules (from specific condition to an immediate action), deliberation rules (to satisfy goals), and social/cognitive rules. There are also constraint rules, which constrain the firing of other rules (constraints on conflicting reaction rules and preconditions on deliberation rules).

**Social System / Static Structure**

The only structure in the agent society is that of groups. Agent groups are used mainly to restrict the extent of broadcasting (see below) and for structuring the agent space (for instance by restricting specific interactions between groups members). Groups are open and dynamic (agents may move in and out of groups), and are organised in a hierarchy of sub-groups. Agent may belong to multiple groups.

Agents are composed of sub-agents, which represent an internal process and relate to a layer (reactive, deliberative, or social). These sub-agents are grouped together to form the reactive, deliberative, and social layers. Interactions between layers can take many forms, and is realised through linking agents – agents that are members of more than one group.

**Social System / Dynamics**

The basic communication mechanism in Concurrent METATEM is broadcasting – messages are sent to the environment and not to a specific destination, and each agent accepts the messages it
finds appropriate. The Interface Definition describes which messages an agent can produce and which messages it accepts.

Following is mapping of the granularity of the model in each of the four representational sub-categories

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static Structure</strong></td>
<td>Open ended</td>
<td>Coarse grain</td>
</tr>
<tr>
<td><strong>Dynamics</strong></td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
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).

Following is mapping of the granularity of the model in each of the four representational sub-categories

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

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static Structure</strong></td>
<td>Coarse grain</td>
<td>Coarse grain</td>
</tr>
<tr>
<td><strong>Dynamics</strong></td>
<td>Coarse grain</td>
<td>Fine grain</td>
</tr>
</tbody>
</table>
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.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual Agent</th>
<th>Social System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Structure</td>
<td>Open-ended</td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Open-ended</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
AUML – the Agent UML; Odell J., Parunak H. and Bauer B.

AUML – the Agent UML (Unified Modeling Language)\textsuperscript{16} 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.

\textsuperscript{16} 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 re-usable 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.

Following is mapping of the granularity of the model in each of the four representational subcategories:

<table>
<thead>
<tr>
<th>Static</th>
<th>Individual</th>
<th>Social</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open ended</td>
<td>Open ended</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamics</th>
<th>Individual</th>
<th>Social</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse grain</td>
<td>Fine grain</td>
<td></td>
</tr>
</tbody>
</table>
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.

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.
3. Organizational – the dynamics of agent interaction is described.

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.

Following is mapping of the granularity of the model in each of the four representational sub-categories:

<table>
<thead>
<tr>
<th></th>
<th>Individual</th>
<th>Social</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Open ended</td>
<td>Coarse grain</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Open ended</td>
<td>Coarse grain</td>
</tr>
</tbody>
</table>
References


Symposium on Methodologies for Intelligent Systems (ISMIS'94), Amsterdam, 378-392, 1994


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


