Requirements Engineering
In
Agent-oriented Software Engineering

Shiva Vafadar
Ahmad Abdollahzadeh Barfourosh

December, 2008
CE/PR/87/01
Table of Content

1. INTRODUCTION.................................................................4

2. RESEARCH APPROACH.........................................................4

3. TARGET JOURNALS AND ConFERENCES..............................5
   3.1. Resource Surveys...........................................................5
   3.2. Resource Journals..........................................................5
   3.3. Resource Conferences.....................................................6

4. CURRENT RESEARCH AREAS................................................7
   4.1. Agents and Multi-agent Systems ..................................8
   4.2. Agent-oriented Software Engineering.............................8
   4.3. Agent-oriented Applications..........................................8

5. AGENT-ORIENTED SOFTWARE ENGINEERING IN MORE DETAILS ...9
   5.1. Requirements Engineering.............................................9
   5.2. Analysis and Design....................................................10
   5.3. Implementation...........................................................10
   5.4. Testing........................................................................11

6. REQUIREMENTS ENGINEERING IN AOSE ACCORDING TO SURVEYS
   11
   6.2. A Survey on Agent-oriented Software Engineering........................12
   6.3. An Overview on Current Trends in European AOSE Research ........12

7. RECENT RESEARCHES ON AGENT-ORIENTED REQUIREMENTS
   ENGINEERING.........................................................................14
   7.1. Operational Modeling of Agent Autonomy: Theoretical Aspects and
        a Formal Language................................................................15
   7.2. Requirements Analysis of an Agent's Reasoning Capability ..............17
   7.3. Goal Oriented Requirements Analysis and Reasoning in the Tropos
        Methodology......................................................................19
7.4. Contributions of Quantitative Modeling to Agent-Oriented Software Engineering

7.5. Incorporating Commitment Protocols in Tropos

7.6. Adaptive Replication of Large Scale Multi-agent Systems

7.7. A Product Line Requirements Approach to Safe Reuse in Multi-agent Systems

7.8. On the Cost of Agent Awareness of Negotiating Services

7.9. Allocating Goals to Agent Roles during MAS Requirements Engineering

8. CONCLUSION

9. FURTHER WORK

10. RESOURCES

10.1. Agents and Multi-agent Systems

10.1.1. Searching

10.1.2. Reasoning

10.1.3. Learning

10.1.4. Environment

10.1.5. Coordination and collaboration

10.1.6. Planning

10.1.7. Organizational structure

10.1.8. Ontology

10.1.9. Cognitive Concepts

10.1.10. Social Aspects

10.1.11. Communication

10.1.12. Negotiation

10.2. Agent-oriented Software Engineering

10.2.1. Requirements

10.2.2. Analysis and Design

10.2.3. Implementation

10.2.4. Testing

10.2.5. Methodology

10.2.6. Modeling

10.3. Agent-oriented Applications

11. REFERENCES
1. Introduction

In this technical report, we present the result of our survey on requirements engineering of agent-oriented systems. In this way, in Section 2, we fully describe our research approach. To follow our research approach, in Section 3 the complete list of our resources is presented. These resources are the surveys, journals and conferences proceedings we use to identify research areas. Section 4 presents the current research areas in distributed artificial intelligence according to our investigation. In Section 5, agent-oriented software engineering which is an important field of research on distributed artificial intelligence is introduced in more details. In section 6 and 7, we focus on requirements engineering as a research area of agent-oriented software engineering. In this way, we highlight the requirements engineering researches introduced by some surveys on agent-oriented software engineering. We also present some recent researches on requirements of agent based systems in more details in Section 7. In Section 8, the conclusion of our research is presented. We specify the future work to continue our research in Section 9. Section 10, shows a complete list of the papers we used in our survey while Section 11 presents the references of this technical report.

2. Research Approach

In this section, we describe our research approach which is applied to prepare this survey on agent-oriented requirements engineering. As the goal of this survey is reporting the state of the art in this field, we divide research scope into two parts; researches have been done on agent oriented software engineering (AOSE) before the year 2005 and those which are related to current years which we mean between 2005 and 2007.

There are some surveys that investigate the state of the state of the art on AOSE in years 2001, 2004. To introduce works that are related to years before 2005, we use existing surveys on AOSE. Therefore, we refer to these papers as the main resources for researches have been done before 2005. The list of these references presented in section 3.1.

For reporting the current researches, we need some trustworthy resources that contain recent studies on AOSE. In this way, some journals and conferences’ proceedings are selected according to their importance and popularity in AOSE society. The main factors for choosing the conferences are their topics of interest, acceptance rate and the number of papers they receive. Since agent-oriented requirements engineering is the main issue of this survey, some special workshops of conferences are chosen which focus on agent-oriented software engineering. The complete list of this journal and conferences is presented in section 3.2 and 3.3.

After collecting the required resources, it is necessary to classify them. By reviewing the abstract of these papers, we fundamentally identify three main areas for researches on agent-oriented Systems; Agent and multi-agent Characteristics, Agent-oriented Software Engineering and Agent-oriented Applications. Each main area is divided into subsections which explain subfields of the main area. The result of this classification is presented in section 4 which introduces the research areas in agent-oriented systems.

Because we have focused on agent-oriented requirements engineering in this technical report, the papers which are related to this field are reported in more details. Sections
6 and 7 describe the researches which specifically concern to requirements engineering. The final step of this research is finding the state of the art and current trends in requirements engineering. To achieve this goal, we analyze the reviewed papers and compare them with agent-oriented software systems requirements. In this way, we keep in mind the research direction in AOSE which have been presented by other researchers. This helps us to have a comprehensive view on requirements of software systems. Section 8 presents the result of this analysis and consequently open areas for research on agent-oriented requirements engineering.

3. Target Journals and conferences
To follow our research approach that has been described in section 2, we use three main resources as references for our survey:

• Agent-oriented survey papers as the source of researches have been done before year 2005.
• Journals concern agent-oriented researches that report current studies on this issue.
• Agent-oriented conferences as other sources to find current researches in his field.

In the following subsections we introduce each resource.

3.1. Resource Surveys
There are some surveys on agent-oriented software engineering that report the state of the art AOSE in the years they published. Therefore, they are significant sources that help us to find previous researches and trends in this field. We use these surveys to report the researches have been done on requirements engineering in agent-based systems for years before 2005. The title and the year of the surveys we employed in this way are:

• A Survey on Agent-oriented Software Engineering (2001)
• An Overview of Current Trends in European AOSE Research (2005)

3.2. Resource Journals
For reviewing and reporting the recent studies on agent-oriented systems, one of our resources is the journals which concern this issue. The following journals are taken into account in this survey:

• Autonomous Agents and Multi Agent Systems
• Engineering Applications of Artificial Intelligence
• International Journal of Agent-oriented Software Engineering

Since we use the journals for reporting current studies, recent issues of the journals are considered. We have considered 46 journal papers in our survey. Table 1 shows an overview on the referenced papers of these journals during the years 2005 – 2007.
Table 1. Overview on Journal Papers

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Papers</td>
<td>16</td>
<td>19</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

3.3. Resource Conferences

As we mentioned in Section 2, the proceedings of some conferences are used as another resource to find current studies on agent-oriented systems. The conferences referred in this report are:

- Autonomous Agent and Multi-agent Systems (AAMAS)
- Autonomous Agent and Multi-agent Systems: Agent-oriented Software Engineering (AOSE) Workshop
- Software Engineering for Large Scale Multi-agent Systems (SELMAS)
- From Agent Theory to Agent Implementation (AT2AI)

We have considered 401 Conference papers in our survey. Table 2 shows an overview on the referenced papers of these conferences during the current years (between 2005-2007).

Table 2. Overview on Conference papers

<table>
<thead>
<tr>
<th></th>
<th>2007</th>
<th>2006</th>
<th>2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAMAS</td>
<td>94</td>
<td>87</td>
<td>97</td>
</tr>
<tr>
<td>AOSE</td>
<td>14</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>AOIS</td>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>SELMAS</td>
<td>-</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>CSICC</td>
<td>15</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

The key factor that we have in mind for selecting conferences is their topics of interest. Since the main issue of this technical report is agent-oriented software
engineering, some special workshops of conferences are also chosen which focuses on AOSE.
Another important factor for choosing these conferences is the number of papers they receive and the acceptance rate of the conference. When the number of papers accepted in the conference is high but the acceptance rate is low, this means that the conference receives a great number of papers on the field and it most likely covers a good range of research on this field.
We also have selected CSICC (computer society of Iranian computer conference) to consider the national research on distributed artificial intelligence as well.

4. Current Research Areas
To follow our research approach (described in Section 2), we reviewed the abstract of the collected resource papers (explained in section 3). The complete list of these papers presented in Section 10)
According to our investigation, we fundamentally identified three main areas in distributed artificial intelligence research; Agents and multi-agent characteristics, Agent-oriented software engineering and Agent-oriented applications.
We classify the resource papers into these three main groups. Furthermore, each group is divided into some sub-areas to precisely specify the research areas each paper belongs to. We call the result of this classification “Research Tree” which is shown in Fig1. Each branch of this tree shows the main issues of researchers’ interest in distributed artificial intelligence according to our viewpoint. In the following subsections we describe each main area briefly.

Fig1: Research Areas
4.1. Agents and Multi-agent Systems

Agents are new entities in artificial intelligence that are the base of each multi-agent system. However they are constructed according to traditional capabilities of artificial intelligent systems, but they also have additional characteristics that make them require specific capabilities. To cover all the required aspects of agents and multi-agent systems a lot of researches have been done on this field. Therefore, agents and multi-agent characteristic field of our research tree contains various areas. During our investigation, we found that many papers are classified in this branch of our research tree.

We have divided this field into 12 sub areas; Learning, Searching, Reasoning, Environment, Coordination & Collaboration, Communication, Social Aspects, Negotiation, Cognitive Concepts, Ontology, Organizational Structure and Planning. Because the focus of this paper is not on this branch of our tree, we are not going to describe its concepts and concerns completely. But the complete list of the papers have been classified in these areas are presented in section 10. We leave preparing the technical report of this part of the tree as further work.

4.2. Agent-oriented Software Engineering

Because agents are new paradigm in building software systems, they require specific approaches for engineering these systems. Especially because agents usually are used in large distributed software systems, specific requirements in software engineering are raised. There are many discussions about agents and why they are different from objects. We refer the interested reader to [JEN2000-a],[ZAM2004],[JEN2000-b].

Agent-oriented software engineering is a powerful way of approaching large scale software engineering problems and developing agent-based systems. This field contains main concepts of software engineering and it concerns special aspects of agent and multi-agent systems. Because software engineering is mainly related to software development, main phases of software development lifecycle are required in agent-oriented software engineering. Therefore, all phases of software development are selected as subsections of this branch of research tree; Requirements Engineering, Analysis and Design, Implementation and Testing.

We also considered the main layers of agent systems such as methodologies and processes, modeling and tools that support agent-oriented software development. As a result, this branch of our research tree contains 7 sub-areas. We describe each sub area in more detail in section 5.

4.3. Agent-oriented Applications

Agents are precious when they are used to produce software applications. All the works have been done by researchers on distributed artificial intelligence have one goal; producing the better agent-oriented applications. Therefore, using the agents to implement the software systems is the interest of both researchers and practitioners. During classifying the papers, we found that a lot of researches have been done on this field and many works have been reported which are related to applications which are using agents.

Because we are not going to concentrate on this field, we just introduce the papers reporting research on applying agent technology for developing software systems.
The complete list of the papers have been classified in these areas is presented in section 10.3.

5. Agent-oriented Software Engineering in more details

Agent-oriented software engineering is a powerful way of approaching large scale software engineering problems and developing agent-based systems. This field contains main concepts of software engineering and it concerns special aspects of agent and multi-agent systems.

During decomposing this branch to related subsections, we try to consider two main perspectives in software engineering. First, because software engineering is mainly related to software development, main phases of software development lifecycle are selected as subsections of this branch of research tree. As a result, four subsections are related to main activities of software development lifecycle; Requirements Engineering, Analysis and Design, Implementation and Testing.

Other perspective states that software engineering is a layered technology [PRE2005]. Consequently, agent-oriented software engineering has various layers as well [AYA2002]. We also take into account this viewpoint for classifying the reviewed papers. Therefore, three subsections are related to layers of software engineering. These sections are Modeling, Methodologies and Tools.

5.1. Requirements Engineering

Understanding the requirements of a problem is among the most difficult tasks that face a software engineer. Requirements engineering helps software engineers to better understand the problem they will work to solve. Requirements engineering addresses the identification and specification of the (functional or non functional) features expected for the system to be developed, abstracting from the manner in which this functionality is realized in a design and implementation of this system.

An agent-oriented view on requirements engineering can benefit from the more specific assumptions on structures and capabilities expected for agents, compared to software components in general. To obtain these benefits a dedicated agent-oriented requirements engineering process can be performed that takes into account specific agent-related structures and capabilities.

In the process of building agent systems, software engineering principles and techniques, such as scenario and requirements specification, analysis, modeling, verification, and validation, can be supported by the reusable results of such a requirements engineering. According to this perspective, requirements are the foundation of all the (agent-oriented) systems which affect the following phases of software development such as analysis and design, implementation and testing.

Some agent-oriented researchers select this area of agent-oriented software engineering as their research subject. The list of the papers of agent-oriented requirements is presented in section 10.2.1. Because the main issue of this technical report is agent-oriented requirements engineering, we fully describe it in more detail in following next sections.

During classifying the reviewed papers, we had in mind different aspects of agent and multi-agent systems requirements; type of requirements, important roles for defining the requirements, requirements engineering phases, methods and models for specifying agent requirements.
5.2. Analysis and Design

At a technical level, software engineering begins with a series of modeling tasks that lead to a specification of requirements and a comprehensive design representation for the software to be built. The purpose of design is transforming the requirements into a design of the system-to-be, evolving a robust architecture for the system and adapting the design to match the implementation environment.

In comparison with conventional entities of software development, agents are more similar to objects but they introduce new concepts of abstraction and new levels of encapsulating. Therefore agent-oriented analysis and design uses different concepts, approaches and method.

An agent-oriented view on software analysis and design encompass different granularity of the system. Therefore, analysis and design of the agent based systems requires specific analyzing and modeling approaches which results agents at design level as building block of the agent based system. Different agent-oriented methodologies present individual approaches for designing the agent based systems which take into account specific agent-related structures and capabilities.

There are three research area related to Analysis and Design. First, the activities, methods, and concepts related to agent-oriented analysis and design. Second, different architectures produced by applying each activity and method at different domains and basic architectures of agents and multi-agent systems. Third, modeling methods and notations specifies the specific concepts of agent-oriented analysis and design.

During reviewing the papers, we classify the two first areas at this section. But those papers which are concern to the modeling aspect of multi agent systems are classified at modeling subsection which is described at section 5.6.

Some agent-oriented researchers are interested in concepts, activities and methods that are related to analysis and design of multi-agent systems. The list of the surveyed papers of this area is presented in section 10.2.2.

5.3. Implementation

Implementation is the process of converting the design to the code which is applicable by machine. Implementation is the phase which realizes all the previous activities of the software development from abstraction to realization. The principles and concepts the coding tasks are closely aliened to programming style, programming languages and programming methods.

Implementing the agent-oriented systems is the process of converting designed agents and multi-agent systems to real running agents by using (usually) object oriented programming languages.

There are some important features at software implementation. One of them is programming language, editors, compilers and all the tools which make it possible to produce a running version of the agent base systems. This is not the subject of this section. We will consider these tools along with the other required tools at the development life cycle at section 6.7.

It is important to consider that implementation is totally different with applications of multi-agent systems. Applications are implemented system of different domains which use agent technology as their base approach to solving problems. These types of papers are classified at a different branch which is named applications.

In this section we concern the papers which are related to programming techniques and methods. Concepts which are related to migrating from design to code, especial
algorithms for implementing the agent based concepts, common frameworks for implementing agent based applications, run time problems of multi-agent systems are the subjects which we classified at this section. The list of the papers we put in this section is presented in section 10.2.3.

5.4. Testing

Software testing is a critical element of software quality and represents the ultimate review of specifications, design and code generation. It acts as a quality provider to other phases of software development life cycle. Testing focuses primarily on evaluating or assessing product quality which is realized through finding and documenting defects in software quality, advising on the perceived software quality, validate and proving the assumptions made in design and requirement specifications through concrete demonstration, validate that the software product works as designed, and validating that the requirements are implemented appropriately. Over the last years, the view of testing has evolved, and testing is no longer seen as an activity which starts only after the coding phase is completed. Software testing is now seen as a whole process that permeates the development and maintenance activities. Thus, each development/maintenance activity should have a corresponding test activity.

As agent-oriented systems are increasingly applied in mission-critical services, assurances need to be given to their owners and users that they operate properly. Therefore, testing takes a more important role in developing agent-oriented system. Consequently, agent-oriented testing becomes an interesting area of research in agent-oriented software engineering. The list of papers on agent-oriented testing is presented at section 10.2.4.

These papers concerns two main approaches on verification; Static and Dynamic verifications. Works on static verification of agent systems are related to model checking. This method is based on examining static models of the product. Researches on agent-oriented model checking tend on formal methods and temporal logic. Methods such as ATL (Alternative-time Temporal Logic), SRML, model checking of continuous flow of instants in the context of epistemic states of multi-agent systems, the model specifications language for the spin model checking system automatically, Modal Logics (such as CTL, CTLK, ATL) and Interpreted Systems are the methods which are concerned at this field.

Researches on dynamic verification of multi-agent systems (MAS) have mainly investigated monitoring MAS at run-time in order to observe abnormal behaviors. Testing the multi-agent systems during different stages of system development such as unit testing, integration testing, system testing are the subjects of the papers of this branch. Many papers of this section try to add testing abilities to the current agent-oriented methodologies.

6. Requirements Engineering in AOSE according to surveys

As we have described in section 2, we review the researches before year 2005 based on the surveys. Following this approach, in this section we present the requirements engineering researches according to the surveys on agent-oriented software engineering.

Wooldridge et al. in [WOO2001] introduces the AOSE state of art in year 2001. In their report, they do not mention any specific research on requirements engineering. But they consider three main roles for formal methods in requirements engineering: specification of software systems, for directly programming systems, and in verification of the systems. According to this view, they report researches related to formal methods on specification. The idea is that such a specification would express the desirable behavior of a system.

In this way they refer to agent theory that explains how mental aspects of agency interact to generate the behaviour of an agent. They consider a temporal modal logic as the most successful approach to formal agent theory.

They introduces the result of the researches have done by Cohen-Levesque in theory of intention [COH 1990], and the Rao-Georff for belief-desire-intention modeling [RAO1995],[WOO2000-a] as two of the best known such logical frameworks.

The key technical problem faced by agent theorists is developing a formal model that gives a good account of the interrelationships between the various attitudes that together comprise an agents internal state [WOO1995].

Comparatively few serious attempts have been made to specify real agent systems using such logics.

6.2. A Survey on Agent-oriented Software Engineering

Tveit in [TVE2001] in year 2001 just introduces [DEP2000] as a research on requirements engineering. According to this report, Depke et al. [DEP2000] apply formal graph theory on requirement specifications for agent-systems in order to maintain consistency when the requirements are transformed into a design model.

He also introduces Gaia [WOO2000-b] and MaSE [DEL2001] methodologies, as method which construct software systems base on predefined requirements. He mentions that Using Gaia, software designers can systematically develop an implementation-ready design based on system requirements. He also states that in MaSE methodology, capturing goals, the first phase, transforms the initial system specification into a structured hierarchy of system goals. This is done by first identifying goals based on the initial system specification’s requirements.

6.3. An Overview on Current Trends in European AOSE Research

Bernon et al in their survey [BER2005] in year 2005 consider requirements analysis as a phase which software functionalities and constraints are defined.

They define this phase as an activity which in the designer collects and analyzes the software requirements, which are usually considered from two perspectives: User and System. The latter is detailed and more technical expression of what the customer specifies in the User Requirements. System Requirements consist of functional (services the software should provide), non– functional (constraints on the services) and domain requirements (coming from the application domain).

In their survey they consider formal and non-formal approaches and multi-views paradigm.
For formal approaches they report the following researches:

- **LORA (Logic of Rational Agents) [WOO2000-a]** which is founded on a first-order logic, includes a BDI (Belief- Desire-Intention) [RAO1995] component (used for the agent architecture), a temporal component (used for specifying the system dynamics), and an action component (used to represent agents’ actions).

- **Situation Calculus [McC1969]** is another expression of this field of research; it is a first-order logic (with some extensions to second-order logic) capable of representing dynamic domains. IndiGolog [DeG2004] is a recent implementation of situation calculus, supporting the high-level programming of robotic intelligent agents that can perform online planning and plan execution in partially unknown environments. In IndiGolog (that is part of the GOLOG [LEV1997] family), environment dynamics is modelled using situation calculus while the agent behaviour is designed in a procedural way.

- **Another formal approach** is due to M. Luck and M. D’Inverno [LUC2001] and it is an application of the Z language [SPI1992] to the specification of agents. Agents in Z are defined within a four-layer hierarchy that includes: entities (inanimate objects with attributes), objects (entities with capabilities), agents (objects with goals), autonomous agents (agents with motivations). In this work the authors take profit of the great number of existing experiences in Z for inheriting a great number of tools that include code production and model checking capabilities. Another approach that uses the Z formalism (and state-charts) can be found in [HIL2000].

They also consider non-formal approaches to the specification of agent systems which are mostly based on the use of structured natural language and graphical notations. They review the specification phase of agent-oriented methodologies and fundamentally identify three categories of non-formal specifications:

- **Functional oriented** [JAC1992] (often adopting use-case diagrams) such as
  - PASSI methodology [COS2005]
  - the ROADMAP [JUA2002], an extension of Gaia [WOO2000-b],

- **Goal oriented approaches** [VAN2001] (that aim at identifying the goals of the system and eventually dividing them among agents) such as
  - MESSAGE [CAI2002],
  - INGENIAS [PAV2005],
  - Tropos [BRE2004].

- **Role-oriented approaches** [KEN2000] (they adopt the role as the key abstraction for specifying a MAS, they are often also concerned about designing roles/agents coordination) such as
  - Gaia [WOO2000-b],
  - SODA (Societies in Open and Distributed Agent Spaces) [OMI2001]
  - RICA (Role/Interaction/Communicative Action) [SER2004]
  - also the cited MESSAGE, INGENIAS, and PASSI.

- **Multi-views, multi-perspectives, multi-level approaches** base their philosophy on three well-known methods for tackling complexity; Abstraction, Decomposition, Hierarchy.
  - The structuring of a multi-agent system in several viewpoints appears in many methodologies such as
    - MAGMA approach [DEM1995]. It considers the five Latin vowels (initially only the first four): Agent, Environment, Interactions, Organization, and User,
MESSAGE [CAI2002] and INGENIAS methodologies [PAV2003], which redefine viewpoints as organization, agent, domain/environment, goals/tasks, and interactions.

- The concept of level in agency is also another way of considering several views.
- It has been initially introduced by Newell [NEW1982] and Jennings [JEN2000] recalled the knowledge level and complemented it with a new social level. The knowledge level is concerned with the agent seen as an asocial problem solver while the social level looks at the agent organization as its main focus.
- Other works in this direction presented different perspectives [COS2002] [COS2004], which are more directed to the representation of the system from a different point of view (architectural, social, knowledge, computer, resource, autonomy) rather than a different level of abstraction.
  - MAS-CommonKADS [IGL1998] (organization, tasks, experience, agents, communications, coordination, and design),
  - ODAC [GER2003], which uses the five ODP viewpoints (enterprise, information, computational, technology and engineering) [ISO1995],
  - MASSIVE [LIN2001] (that includes seven views: environment, task, role, interaction, society, architectural, system).

Furthermore, for requirements elicitation, INGENIAS proposes to base on Activity Theory to analyze intentional and social issues of the system, by providing a set of contradiction patterns that guide the developer in the identification of conflicts in the specification about the agent and the organization goals [FUE2004].

### 7. Recent Researches on Agent-oriented Requirements Engineering

In this technical report we are going to find the recent issues on requirements engineering in agent-oriented software engineering. To follow our research approach we review the research of current years which are focused on requirements engineering. In this review, we concentrate on

- the problem which was the motivation of the researchers
- the activity related to requirements engineering which the paper focuses on (e.g. elicitation, analysis, specification, verification or validation and change management)
- The contribution of the paper
- The research method has been used to achieve the results
- The case study has been presented to explain the concepts and results
- The agent-oriented software engineering methodology has been used during the development of the case study
- The tools has been used during system development (in each phase of software development)
- The results have been achieved by the paper and especially the way the results have been presented (e.g. quantitatively or qualitatively, in comparison with other related papers, based on the experience during the developing case study)
- Possible further works on the research presented at the paper
Using this frame to summarize the papers gives us a framework to evaluate the state of the art in this field of research. In the following sections we present each paper which is related to requirements of agent based systems in recent years and we have considered in our survey.

7.1. Operational Modeling of Agent Autonomy: Theoretical Aspects and a Formal Language

Weiss et al. in [WEI2005] consider autonomy as an intelligent agent capability and show how a requirements analysis of an agent’s autonomy capability can be made.

**Problem:**
The problem they have encountered in agent-oriented software system is that although an agent should have autonomy and act as an autonomous entity but it will always be the designers or users who are responsible for its actions in a legal sense. Hence, the implementation of autonomy as an agent property will require rigorous modeling and verification, so as to ensure maximum dependability of the systems acting autonomously. Without this dependability, it is unlikely that autonomously acting agents will be broadly used in industrial, commercial and scientific applications.

**Research method:**
To specify autonomy precisely, in this paper autonomy is defined as the degrees of freedom left to the agents for the execution of activities. The basic view underlying this research is that agents are embedded in a social frame that regulates their behaviour. This social frame, called role space, is composed of a set of roles which are available to the agents and through which they can try to achieve individual and joint objectives. Roles serve as a means for specifying desired behaviour and for achieving behavioural predictability, but not to make sure that agents never exhibit unexpected and undesirable behaviour (which would simply be impossible if autonomy is taken seriously).

They define roles consists of set of activities to which sanction and norms are attached. They distinguish between three different types of norms (namely permissions, obligations, and interdictions) and two types of sanctions (reward and punishment). While norms correspond to behavioural expectations held by agents against each other in their capacities as role owners, sanctions denote (potential) consequences of norm-conforming and norm-violating behaviour. Hence, through norms and sanctions, a system designer can explicitly specify the limits within which an agent is supposed to act autonomously, and how these limits are enforced.

**Contribution:**
In this paper an autonomy specification language (ASL) is introduced that allows for a precise specification of the activities to be carried out by a set of agents, the deontic constraints imposed on these activities, and the implications of activity execution on particular constraints. ASL allows for the automatic detection and handling of norm conflicts, such that conflicts can either be resolved at design time or appropriate measures can be taken for their runtime settlement.

**Summary of the method:**
The most general abstraction employed by ASL is that a role space composed of several roles to be played by the individual agents in their attempt to achieve their goals. Therefore, role space spec is defined as a production rule. In ASL, each role is defined through a set of characteristic activities. Attached to each activity of each role is at least one status statement that specifies the norms and sanctions an agent playing the role is exposed to with respect to this particular activity. ASL distinguishes three types of norms – permission (indicated by the keyword p), obligation (o), and interdiction (i) to carry out the activity – and two types of sanctions – reward (re) and punishment (pu) – that apply in the case of norm conformance and norm deviation, respectively. Agents may ignore or violate norms, be it intentional or not. ASL takes care of this fact by enabling designers to explicitly specify the consequences of norm-conforming and norm-deviating behaviour in terms of positive and negative sanctions (i.e., reward and punishment). In other words, norms alone do not impose any limitations on possible agent behaviour (since this is impossible due to our definition of autonomy), they rather work indirectly via the agent’s internal reasoning about the attached sanctions, making certain behaviours (which may be undesirable from the designer’s point of view) undesirable for the agent. To support this idea ASL defines norm-sepc, sanction-spec in addition with two different statements: independent status statement, dependent status statement. It also introduces syntax for four different types of activities, called basic, activating, deactivating and request activities. This syntax enable system analyst to explicitly capture adaptive autonomy and meta-autonomy and requests for behaving cooperatively with other agents.

Methodology:
In this paper requirement specification is not attached to any particular methodology. However since it is based on the roles, it may be used in many existing agent-oriented methodologies.

Case Study:
In this paper an agent-based electronic supply chain management system (“eSUPPLY”) has been selected as a case study to explain different concepts and proposed notation by the ASL.

Tools:
They do not introduce any tool to support the language that they have introduced.

Results:
ASL offers two main benefits. First, it is a highly expressive language that enables designers to specify agent autonomy at a very precise level. Consequences of both norm-conforming and norm-deviating behaviour can be captured by means of positive and negative sanctions. Second is that it allows for the detection and resolution of autonomy-induced conflicts already at design time. These results have been presented according to comparison subjectively with other modeling languages according to the features each one supports.

Further Work:
Researchers present three main areas as the shortcoming of their current work and as their plan for further work.
First, the usual constructs for role modelling (inheritance, composition, etc.) and assignment to individual agents and modeling complex structural relations between roles

Second, introducing explicit time and hence allow for the specification of deadlines as temporal constraints on norms (i.e. the time interval between a request, the execution of the corresponding activity and the initiation of a possible sanction) or other temporal aspects of autonomy (e.g. norms that are valid only at a certain time).

Third, giving a formal (e.g. possible worlds) semantics to ASL to provide a proper theoretical grounding and ultimately pave the way for model checking the autonomy-related properties of a system.

7.2. Requirements Analysis of an Agent's Reasoning Capability

Boss et al. in [BOS2005-a] consider reasoning as an intelligent agent capability and show how a requirements analysis of an agent’s reasoning capability can be made.

Problem:
In the literature, software engineering aspects of reasoning capabilities of intelligent agents have not been addressed well. Some literature is available on formal semantics of the dynamics of non-monotonic reasoning processes. However, these approaches focus on formal foundation and are far from the more practical software engineering aspects of actual agent system development.

Research method:
In this paper, a number of scenarios of practical human reasoning processes considered as ‘reasoning by assumption’ have been analysed and specified to identify requirements that are characteristic for this reasoning pattern.
In fact, this paper is the continue of their previous work on Analysis of Design Process Dynamics [BOS2004] and formalization and analysis of Human Reasoning Traces[BOS2005-b].
Required dynamic properties at different levels of aggregation (or grain size) have been identified. Logical relationships have been determined between dynamic properties at one aggregation level and those of a lower aggregation level. In this paper, these characterising properties have been formalized using the temporal trace language TTL, thus enabling automated support of analysis.

Contribution:
Dynamic properties have an important role to specify (both functional and non-functional) requirements of agent systems [BOS2004]. Thus, the contribution of this paper is in the method to requirements analysis of an agent’s reasoning, and the reusable results obtained by that method. In this paper, it is shown for reasoning by assumption how relevant dynamic properties at different levels of aggregation can be identified as requirements that characterise the reasoning capability.
Summary of the method:
For reasoning processes in natural contexts, which are usually not restricted to simple deduction, dynamic aspects play an important role and have to be taken into account, such as dynamic focusing by posing goals for the reasoning, or making (additional) assumptions during the reasoning, thus using a dynamic set of premises within the reasoning process.
Reasoning processes or their outcomes cannot be understood, justified or explained without taking into account these dynamic aspects. Therefore, the approach to the semantically formalisation of the dynamics of reasoning is based on the concepts reasoning state, transitions (which specify allowed transitions) and traces (which are sequences of reasoning states such that each pair of successive reasoning states in such a trace forms an allowed transition).
To specify properties on the dynamics of reasoning, the temporal trace language TTL is used and reasoning state, traces and dynamic properties are specified according to the agent's ontology by TTL.
Dynamic properties includes three types of properties.

- Local properties which address the step-by-step reasoning process of the agent such as assumption initialization, prediction effectiveness, observation initiation effectiveness, observation result effectiveness, evaluation effectiveness, assumption effectiveness, assumption persistence, rejection persistence, observation result persistence.
- Intermediate properties which are properties at an intermediate level of aggregation, such as proper rejection grounding, prediction-observation discrepancy, implies assumption incorrectness, observation result correctness, incorrect prediction implies, incorrect assumption which are used for the analysis of global properties.
- Global properties which address the overall reasoning behaviour of the agent such as reasoning termination, correctness of rejection, static world, domain knowledge correction and world consistency.

A number of logical relationships have been the identified between properties at different aggregation levels as a AND-Tree. According to this tree, it is possible to define grounding variables which make a specification of local properties more complete by stating that there is no other means to produce certain behaviour. The relationships depicted in AND-tree should be interpreted as semantic entailment relationships. Logical relationships between dynamic properties can be very useful in the analysis of empirical reasoning processes.

Methodology:
In this paper requirement specification is not attached to any particular methodology. However a design of an existing software agent performing reasoning by assumption was analysed and this agent was designed using the component-based design method DESIRE. But they are going to consider some methodologies and adapt their work with the concepts of the methodology.

Case Study:
However in the paper results have been presented based on a large number of such checks which have indeed been performed for several case studies in reasoning by assumption. But the case study hasn't been introduced explicitly.
For explaining the concepts and introduced properties an example of reasoning by assumption has been used.

**Tools:**

To generate simulation traces, local dynamic properties were used by a software system which is called LEADSTO. LEADSTO is a Language and Environment for Analysis of Dynamics by SimulaTiOn which has been developed by authors in their previous works. Using such traces, the requirements engineers and system designers obtain a concrete idea of the intended flow of events over time. In addition, a special tool has been developed that takes a formally specified property and a set of traces as input, and verifies whether the property holds for the traces. Using this checker tool, dynamic properties (of all levels) can be checked automatically against traces, irrespective of who/what produced those traces:

**Results:**

A large number of checks have been performed for several case studies in reasoning by assumption. The results of the simulation traces for different types of attributes have been presented in this paper, are compared to the human traces and prototype resources which have been presented in their previous works. The base of this comparison has not been described in the paper. But it seems that it is based on the number of scenarios have been evaluated during evaluation.

**Further Work:**

The authors present their future work as exploring the possibilities to incorporate the approach based on dynamic properties presented in the paper within architectures such as Tropos [BRE2004], KAOS [DAR1998] or GBRAM [ANT1996]. Especially for architectures that provide a specific language for formalisation of requirements (KAOS for example uses a real-time temporal logic to specify requirements in terms of goals, constraints and objects), these possibilities are promising.

7.3. Goal Oriented Requirements Analysis and Reasoning in the Tropos Methodology

Giorgini et al. in [GIO2005-b] propose a formal goal model to concrete requirement analysis phase of Tropos methodology.

**Problem:**

The problem they have encountered in Tropos methodology is that although a strategic dependency model provides hints about why processes are structured in a certain way, it does not sufficiently support the process of suggesting, exploring and evaluating alternative solutions. At the other hand, approaches presented in the literature for modeling and analyzing goals do not work for many domains where goals can not be formally defined and the relationship among them can not be captured by semantically well-defined relations such as AND-OR ones.
Research method:
In this paper, the results of two separate works have been combined. This paper adopts a formal goal model defined and analyzed in [GIO2002] to make the goal analysis process of Tropos methodology concrete through the use of forward and backward reasoning for goal models.

Contribution:
The contribution of this paper is to enrich the requirement analysis phases of Tropos methodology by adding a formal goal modeling to these phases. This goal model supports both qualitative and quantitative relationships between goals and can be used to find that if given a goal model, and assuming that certain leaf goals are fulfilled, are all root goals fulfilled as well. And given a goal model, find a set of leaf goals that together fulfill all root goals. This extension allows the software engineer to cope with qualitative relationships and inconstancies among goals. It also adds some goal reasoning tools which help the developer of Tropos models analyze them automatically and make sure they are consistent with developer intuitions.

Summary of the method:
In Tropos, a crucial role is given to early requirements analysis that precedes prescriptive requirements specification for the system-to-be. Agent, goal and other mentalistic notions are used to support all software development phases, from early requirements analysis to implementation. By so doing, one can relate the functional and non-functional requirements of the system-to-be to relevant stakeholders and their intentions. Software development begins by identifying relevant stakeholders (represented as actors) and their goals. These root goals are analyzed, refined, and delegated to existing on new actors. The system-to-be and its components come about as new actors who are responsible for the fulfillment of some of the original or refined goals. The whole process ends when sufficient goals have been delegated so that if all actors fulfill their responsibilities, all root goals are fulfilled.

The adapted goal model in [GIO2002] helps requirement engineer to analysis the goals and find if the goals captured in the requirement phase are consistent. This goal model applies goal graph and goal relation notations to specify goals and their relations formally. The introduced notations describe the relation between source goals and target goals and how they affect each other. This means that if the (fully or partially) satisfaction of the source goals results (fully or partially) satisfaction of the target goal and if the (fully or partially) denying the source goal will cause (fully or partially) denying the target goal and vice versa. A complete set of relation between the source goals and target goals are defined.

Given a goal graph and an initial values assignment to some goals (typically leaf goals), forward reasoning focuses on the forward propagation of these initial values to all other goals of the graph according to the rules defined in the model. Initial values represent the evidence available about the satisfaction and the denial of a specific goal, namely evidence about the state of the goal. The model also admits conflicting situation in which there are both evidence for satisfaction and denial of a goal.

Forward reasoning is adopted in Tropos for evaluating the impact of the adoption of the different alternatives with respect to the non functional requirements of the system.

In backward reasoning, the desired final values of the target goals are set and possible initial assignments to the input goals are found which would cause the desired final values of the target goals by forward propagation. Using his method, it is possible to
fin among the alternatives of the goal model those with the minimal cost that allow the system analyst to obtain the desired goal.

These methods and concepts have been described in the using the introduced case study.

**Methodology:**
This paper focuses on Tropos methodology and its requirement analysis phases

**Case Study:**
In this paper an agent-based electronic media shop system (“Medi@”) has been selected as a case study to explain the requirement analysis phases of Tropos methodology and different concepts and proposed notation of goal modeling. Media Shop is a store for selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like. Media Shop customers (on-site or remote) can use a periodically updated catalogue describing available media items to specify their order. Media Shop is supplied with the latest releases from Media Producer and in-catalogue items by Media Supplier.

**Tools:**
Forward and backward reasoning is supported in Tropos by the goal reasoning tool (GR-Tool). Basically, the GR-Tool is graphical tool in which it is possible to draw the goal models and run the algorithms and tools for forward and backward reasoning. For the backward reasoning a tool called GOALSOLVE is presented in the paper. GOALSOLVE takes as input a representation the goal graph, a list of desired final values and, optionally, a list of user desiderata and constraint and a list of goals which have to be considered as input.

**Results:**
In this paper, just the results of the forward and backward reasoning in the case study have been reported for three different goal set selections. Because this paper is a research to adopt the goal modeling method in Tropos methodology, there is not any related work to compare the results with.

**Further Work:**
Authors envision other types of formal analysis for Tropos models. In particular, working on formal actor dependency models and develop scalable and usable social analysis techniques that complement the temporal and intentional analysis techniques.

### 7.4. Contributions of Quantitative Modeling to Agent-Oriented Software Engineering

Brinn et al. in [BRI2005] believe that quantitative modeling is a requirement of developing multi-agent systems and agent-oriented methodologies should consider it as a requirement during developing these systems.

**Problem:**
The authors of this paper believe that many approaches to characterizing the behavior of MAS have emphasized their qualitative or logical properties, and have been accompanied by software development methodologies that support making such characterizations. However, many key attributes of MAS are quantitative in nature.
Much work has been done in the agent-oriented software engineering (AOSE) community to characterize qualitative properties of MAS. Less work has been done in AOSE towards asserting quantitative claims about MAS. This is likely due to the intrinsic difficulty in making quantitative claims about systems whose behavior is often highly nondeterministic.

**Research method:**
In this paper, the results of developing a large scale multi-agent system and the challenges developers were faced to develop such system have been reported.

The challenge posed by DARPA’s UltraLog program was to assure the survivability of a complex, large, and dynamic MAS. The stated goal for this system was to “operate [the application] with up to 45 percent information infrastructure loss with no more than 20 percent capabilities loss or 30 percent performance loss in an environment characterized by wartimes loads and directed adversary attack.” In order to validate any claims of meeting these clearly numeric measures of success, accurate measures of these characteristics in a running system were required.

According to their experience in developing the DARPA UltraLog program, authors report potential benefits of quantitative modeling for various purposes.

**Contribution:**
Based on their experiences in UltraLog, the authors suggest that quantitative modeling, simulation and analysis can make valuable contributions to agent-oriented software engineering. To that end, they recommend that supporting such modeling should be an explicit design step, and such simulation and optimization be an explicit part of the configuration process. We suggest two approaches towards increasing the role of quantitative modeling in AOSE.

**Summary of the method:**
In this paper authors believe that the quantitative modeling of multi-agent system requirements is a necessity. They consider intrinsic difficulty in making quantitative claims about system whose behavior is often highly non-deterministic as a barrier. However the indicates that the potential benefits of being able to make and support quantitative claims regarding MAS are significant.

They divide quantitative modeling in three main dimensions (Inputs, resources and outputs). They also consider three modes to assert the quantitative modeling (Average case, worst case and best case).

They also mention factors make creating such assertions for MAS difficult:

- The complexity of composed distributed systems and the unpredictability of chaotic emergent behavior
- The component models themselves are often elusive
- The goal of quantitative models is to be able to abstract the essence of a complex system to allow for efficient analysis by simulation and experimentation. Yet even using code as the ultimate (if slower) model of itself, the non-determinism of the system makes such analysis difficult or unreliable.

The potential benefits of quantitative models are considered as the ability to predict System functionality, Hardware configuration, Software configuration and External environment.
The authors report their experience in developing a large scale multi-agent system and how as the program progressed the use of quantitative models become increasingly essential in many aspects of their approach such as defense coordination, dynamic control, load balancing and testing. They mention that the key lesson they have learnt during the developing their system was the need to reduce or eliminate non-determinism where possible, to ensure reliable results and have a measure to improve the system.

They report several lessons learnt during their experience as:

- Our use of qualitative models was more ‘narrow and deep’, that is, focusing on assessing or protecting particular system attributes, rather than focusing on overall system function. Our use of quantitative models was more ‘broad and shallow’, so that aggregate macro system function was assessed without emphasis on internal state or processing.
- Imposing a quantitative set of requirements by which to judge system performance and survivability led to a need to minimize variability in system performance and to root out sources of non-determinism where possible. While perhaps counter-intuitive, this kind of local sacrifice for larger MAS success has its own AOSE implications.
- Ultimately, it proved difficult to create reliable quantitative models of the UltraLog prototype. Modelability would seem to be a desirable attribute to include from the start, not readily captured after design and development. Many components had inherent non-determinism in their implementations, or reflected multiple operating modes. In particular, components often didn’t have ‘anytime’ modes by which we could moderate the trade-off of time and quality.

They also discuss some suggestions toward increasing the role of quantitative modeling in AOSE.

- In designing agents (and components, for component-based agent architectures), it is important to consider the quantitative aspects of the semantics of the component no less than the component itself.
- To treat each component in a MAS as a functional “black-box.” If you can produce quantitative models of the quality (rate, degree of accuracy, etc) of the output for given inputs (a family of curves, for example), then these models can be used in combination to produce overall models of the entire MAS. Ideally, these performance curves may be experimentally determined, or even specified as requirements a priori

Methodology:
Using specific methodology during system development has not been reported in the paper. However, the authors believe that existing agent-oriented methodologies do not pay enough attention in qualitative modeling of requirements. They propose that including quantitative specification and modeling should be consider as an essential part of the MAS design methodology.

Case Study:
The case study used in this paper is UltraLog which is an application that plans and simulates military logistics operations. The MAS consists of 1000+ autonomous agents, each representing a combat or logistics organization. As the system receives operational plans, combat organizations compute demands for particular quantities of goods, and the logistics organizations determine how they can best satisfy these
demands. The resulting logistics plan contains 25,000 individual plan elements representing demand and transportation for more than 34,000 entities of more than 200 types of major end items. The demand covers ammunition, fuel and food; the transportation covers end-to-end movement schedules for personnel, unit equipment, and ammunition resupply. Once the operation is planned, we simulate its execution, replanning to changes in requirements and deviations between expected and observed behaviors. The network of relationships among agents was dense, with some links static and others dynamic. UltraLog was built using the Cougaar agent architecture [BRI2004] and contained some 10,000+ component or plugin instances.

**Tools:**
In this paper, the ACME test framework (http://cougaar.org/projects/acme) in UltraLog was used to validate and re-compute quantitative models and assess quantitative performance in scripted stress environments.

**Results:**
In this paper, the experience of the authors in developing the Ultra system has been reported and they explained how using the quantitative modeling helped them to achieve the overall quantitative requirement of the system. No result of the system have been reported in comparison with other works or in comparison with not using the quantitative modeling.

**Further Work:**
Authors believe that the role of quantitative modeling in AOSE is clearly worth further research and exploration to improve the reliability and quality of MAS development and deployment. They particularly introduce the further work as:
- augmenting the Cougaar architecture and associated test frameworks to generate, police and validate quantitative models
- a language to capture quantitative aspects of MAS performance
- to add to the expressivity of quantitative models and to connect such expressions with alternative qualitative descriptions
- Investigating a unified model that supports both qualitative and quantitative properties since there are a complex interplay between the qualitative and quantitative aspects of distributed MAS

**7.5. Incorporating Commitment Protocols in Tropos**
Mallya et al. in [MAL2005] consider agents interactions as an important aspect of multi-agent systems. They believe that agent-oriented methodologies don’t pay enough attention on this requirement of agent based systems at the requirement analysis phase. But they accept commitment protocol as a powerful tool to declaratively capture interactions among business partners, thus facilitating flexible behavior and a sophisticated notion of compliance.

**Problem:**
Tropos models dependencies among stakeholders well and accommodates their evolution as the goals and plans of the stakeholders are refined. The requirements serve as reminders and guards throughout the development process. However, Tropos
does not capture agent interaction requirements in the early stages. Protocols are not identified until the detailed design stage whereas dependencies are defined early. Protocols evolve as the design progresses. Tropos can benefit from an interaction model that allows interactions to be refined with each successive stage of software development.

On the other hand, the theory of commitment protocols does not address how interaction protocols and the contexts of their application can be identified in a multi-agent system. Tropos can provide cues for identifying protocols because it identifies actors, their goals, their plans to achieve goals and dependencies.

**Research method:**
This paper synthesizes two trends in the engineering of agent based systems; modern agent-oriented methodologies and commitment protocols declaratively capture interactions among business partners. It considers each method strength and weaknesses and combines these two directions to enrich both of them.

**Contribution:**
The contribution of this paper is to both Tropos and the theory of commitment protocols. Through protocols, this approach gives interactions the same status as goals in Tropos. Interactions among independent parties can be captured early and successively refined based on a theory of protocol subsumption. Because of its identification of stakeholders and their goals, dependencies, and plans, Tropos provides a valuable approach in which to identify and refine commitment protocols.

**Summary of the method:**
In Tropos, a plan is a sequence of steps that an actor may take in order to achieve a certain goal, and a goal is a state which the actor wants to bring about. Plans are means to achieve goals. Plans are executed, goals are achieved, and resources are made available. Nine types of dependencies can exist between actors in Tropos, since dependums on the dependee's side and reasons on the depender's side can be either a plan, a hardgoal, or a resource.

The paper presents three observations about the operational behavior of the dependencies.
- Observation 1 The reason of a dependency cannot be executed to completion, achieved, or made available till its dependum is executed (at least partially), achieved, or made available.
- Observation 2 A dependency's reason is an actor's local view of an interaction protocol.
- Observation 3 Outgoing dependencies can be propagated up the hierarchy in *AND*-decomposition trees.

The paper also provides guidelines for introducing protocols into Tropos using dependencies among actors as the basis.
- Guideline 1: A protocol is required between two actors if and only if at least one dependency exists between them.
- Guideline 2: Protocols cannot realize dependencies that have softgoals as dependums or reasons.
- Guideline 3: If the means for an end are reasons for dependencies, those means should either be parts of local views of different runs of the same
protocol or parts of local views of different protocols that achieve the same interaction. This guideline applies to OR decompositions as well.

- Guideline 4: If the non-root elements of an AND decomposition are reasons for dependencies, those elements should be parts of the local view of the same protocol.
- Guideline 5: If the AND decomposition tree has dependencies, either incoming or outgoing, with two different actors, a 3-party protocol exists between them.
- Guideline 6: If a resource belonging to one actor is the dependum for a dependency with a second actor and a reason for a dependency with a third actor, a 3-party protocol exists between the actors.

Methodology:
Authors claim that for concreteness, they choose the Tropos methodology, which is strong in its requirements analysis, but their results can be ported to other agent-oriented methodologies.

Case Study:
The case study used in this paper is eCulture System for the Trentino provincial government (called PAT). This system provides information about cultural services such as museums to citizens and tourists. There are four stakeholders (top-level actors) in the eCulture System: Citizen, PAT, Visitor, and Museum. These actors have the goals get cultural info, increase Internet use, enjoy visit, and provide cultural services, respectively.

Tools:
Using Tools has not been reported in the paper.

Results:
Results of the method have not been reported in the paper.

Further Work:
In this paper, a protocol-design methodology has been proposed based on hardgoals and the plans that achieve them. Refinement of these protocols should be based on the softgoals of the participants. In this regard, softgoals are analogous to the private policies of a protocol-participant. Developing this line of research is considered in future work.

7.6. Adaptive Replication of Large Scale Multi-agent Systems

Guessoum et al. in [GUE2005] consider fault tolerance as an inevitable issue for large-scale multi-agent systems because of the possibility of partial failure of distributed multi agent systems. In their paper, they apply the fault tolerant methods of distributed systems on multi-agent systems and propose an approach for fault-tolerance of multi-agent systems. The starting idea is the application of replication strategies to agents, the most critical agents being replicated to prevent failures. They describe an approach and related mechanisms for evaluating the criticality of a given agent (based on application-level semantic information, e.g. interdependences, and also system-level statistical
information, e.g., communication load) and for deciding what strategy to apply (e.g., active replication, passive) how to parameterize it (e.g., number of replicas). To gather required information they monitor multi-agent system architecture and individual agents. Agent-monitors use two criteria to evaluate the proposals: communication time between the two hosts (its host and the participant one) and resource cost. Agent-monitors may use several strategies to select a proposal. For example, they may classify the different proposals according the communication time with the corresponding host. Three classes can be considered: low, medium and high. Thereafter, depending on their budget, they choose the most expensive resources. The utilization of these two criteria allows replicating to most reliable hosts.

Problem:
The possibility of partial failures is a fundamental characteristic of distributed applications. New cooperative applications, e.g., air traffic control, cooperative work, and e-commerce, are much more dynamic and large scale. It is thus very difficult, or even impossible, to identify in advance the most critical software components of the application. Furthermore, criticality can vary over run time. The roles and relative importance of the agents can greatly vary during the course of computation, of interaction and of cooperation, the agents being able to change roles and strategies.

Research method:
In this paper, two different middleware, DIMA and DarX have been integrated to build a fault tolerant multi-agent platform. It also introduced two algorithms for computing the dependencies between agents. This information is used to analysis of an agent criticality for defining its importance and the influence of its failure on the behavior and reliability of the multi-agent system.

Contribution:
The contribution of this paper is to give the capacity to the multi-agent system itself to dynamically identify the most critical agents and to decide which fiabilisation strategies to apply to them. This is analog to “load balancing” but for fiabilisation. They use an architecture to automatically and dynamically apply fiabilisation (mostly through replication mechanisms) where (to which agents) and when they are most needed. To guide the adaptive fiabilisation, various levels of information at system level are used like communication load, and application/agent level, like roles or plans.
This paper presented a new approach to make large-scale multi-agent systems reliable. This approach is based on the concepts of interdependence, an agent criticality relies thus on it interdependences with other agents. The agent criticality is then used to replicate agents in order to maximize their reliability and availability based on available resources and their costs.

Summary of the method:
In most existing multi-agent architectures, the observation mechanisms are centralized. In this paper a distributed observation mechanism has been proposed to improve its efficiency and robustness. This mechanism relies on a reactive agent organization. Theses agents have two roles:

- They observe the domain agents and control their replication,
• They build global information and minimize communication.
These two roles are assigned to two kinds of agents: domain agent monitors (named agent-monitors) and host monitors (named host-monitors).
Interdependence graphs were used to describe the interdependences of agents in a multi-agent system.
An adaptation algorithm gives an outline of the adaptation mechanism of the interdependence graphs. The adaptation algorithm is thus used by each agent-monitor to manage the associated node.
In this paper two algorithms have been proposed to compute the interdependence between two agents. The first one considers only the number of messages exchanged by agents and the second one deals with speech acts (performatives).
The analysis of an agent criticality allows defining its importance and the influence of its failure on the behavior and reliability of the multi-agent system. In this paper, authors propose to use the interdependences of agent to define its criticality.

Methodology:
Using specific methodology hasn't been reported at the paper.

Case Study:
The case study has not been introduced at the paper.

Tools:
Using tools has not been reported in the paper. Just DIMA and DarX was mentioned as middleware to build a fault tolerant multi-agent platform.

Results:
Results to validate DimaX ( the proposed architecture) has not been reported in the paper.
But the results of the experiments to measure the monitoring cost have been reported in three cases:
1) a multi-agent system without monitoring,
2) a multi-agent system with monitoring based on algorithm 1
3) a multi-agent system with monitoring based on algorithm 2.
The monitoring cost is thus almost a constant function. It does not increase with the number of agents. The monitoring activity does not increase when the number of agents (domain agents and associated monitoring agents) increases. That can be explained by the proposed optimization in the multi-agent architecture such as the hierarchical organization of monitoring agents and the communication between the agent-monitors and host-monitors.

Further Work:
More experiments with a large-scale real-life application are needed to validate the proposed approach and to analyze the proposed algorithms.

7.7. A Product Line Requirements Approach to Safe Reuse in Multi-agent Systems

Dehlinger et al. in [DEH2005] consider the role of requirements specification in software reuse as a technology to reduce the time and cost of multi-agent software
development. They believe that existing agent-oriented methodologies have failed to adequately capture reuse potential. In their paper, they propose an extensible agent-oriented requirements specification template for distributed systems that supports safe reuse for mission critical multi-agent systems.

**Problem:**
One of the main goals of AOSE was to provide methodologies for reusing and maintaining agent-based software systems. In spite of this goal, AOSE methodologies have failed to adequately capture the reuse potential. A few attempts have been proposed for reuse in an agent-oriented development environment. However, in each case, reuse is positioned in the later stages of design and development.

**Research method:**
The research presented here reaches into three distinct areas of software engineering: agent-oriented software engineering (AOSE), software product-line engineering and software safety.
This paper adapted portions of the Gaia agent-oriented methodology and integrated them with a product-line-like approach to support the safe reuse of the derived requirements of an agent-based, distributed system.
In this way they adopt a product-line-like approach and extend Gaia methodology role schema to provide required information for agent reuse. A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission.

**Contribution:**
The contribution of this paper is that it provides a requirements specification template so that the dynamically changing configurations of agents can be captured and reused for future similar systems. As a result, it provides an extensible agent-oriented requirements specification template for distributed systems that supports safe reuse.

**Summary of the method:**
In order to be able to reason about reuse and system evolution they define a two phase approach. At the first phase, variation points are defined for each role to provide required information. Variation points give a classification of the different levels of intelligence that the role can adopt during its lifetime. An agent’s configuration includes the current set of protocols, activities and permissions that it obtains from a role at its current variation point. A configuration change occurs when an agent’s role variation point changes. Thus, as a role is promoted to a higher intelligence level the configuration of the agent dynamically changes by incorporating additional protocols, activities, permissions and/or responsibilities. The reverse would occur when a role is demoted from a higher intelligence level to a lower intelligence level. Further, for every variation point identified, a binding time is defined which defines the time at which the variation point could be assumed by a role. Potential binding times include specification-time, configuration-time and run-time. They introduce a Variation point schemata to capture requirements and document them. To do this, they present a step by step procedure.
At the second phase that is started upon completion of the initial requirements analysis and development of an agent based distributed system, derived requirements specification are utilize to instantiate a number of members of the distributed system. Since the prior steps have specified all the possible variation points of the roles in the
schemata, a new member to be added is instantiated to the distributed system by specifying each new member to be deployed in the Agent Deployment Schema. This schema has been presented in the paper because Rather than repeatedly defining the requirements of a role for any given agent, the agent deployment schema allows simply defining the intelligence level it can assume. To clarify this step, a process has been explained in the paper.

To both address the need for a software safety analysis mechanism and to exploit the reuse potential within the requirements specification template a tool and a process have been introduced. This process produces an updated Software Fault Tree (SFT) within PLFaultCAT tool such that SFTs can be automatically generated using the new variation point and the previously documented variation points.

Methodology:
The approach maintains consistently with Gaia methodology. This paper adapted portions of the Gaia agent-oriented methodology and integrated them with a product-line-like approach to support the safe reuse of the derived requirements of an agent-based, distributed system.

Case Study:
The case study which is used in this paper is an open, agent-based implementation of the TechSat21 (Technology Satellite of the 21st Century). TechSat21 is a mission designed to explore the benefits of a distributed approach to satellites employing agents. TechSat21 is a constellation (i.e., cluster) of context-aware microsatellites (weighing under 100 kilograms) in which new microsatellites will be deployed to the constellation in phases. New microsatellites potentially have additional capabilities not found in previously deployed microsatellites while sacrificing functionality found in other microsatellites. Within the TechSat21 constellation, each microsatellite must know its context to meet safety requirements placed upon the entire constellation (for example, each microsatellite must know its position in relation to others to avoid collisions). Similarly, microsatellites within the constellation must cooperate to meet mission requirements which may be safety related.

Tools:
The approach presented here maps readily into an existing tool-supported safety analysis technique which is called PLFaultCAT. PLFaultCAT is a product line software fault tree analysis tool which has been reported by the authors. They have reported their work in [DEH2006]. The PLFaultCAT tool can also accommodate the requirements of an agent-based, distributed system documented in the Role Schema and the Role Variation Point Schemata. The Safety Properties (found under the Responsibilities section of the Role Variation Point Schema) help guide the safety analysis.

Results:
In this paper, applying the approach on the case study has been explained. But the results have not been reported or compared with other methods.

Further Work:
Planned future work includes an investigation of how feature dependencies can be better handled within the established framework.
7.8. On the Cost of Agent Awareness of Negotiating Services

Giovanichi et al. in [GIO2005-a] focus on non functional requirements such as time and computational cost and the way agent technology affects them. They believe that it is necessary to assess the computational cost added by agent technology in agent-oriented applications so that we can diagnose the improvements required by state-of-the-art agent technology.

Problem:
While a significant number of agent-based applications for electronic commerce have been presented to the agent community during the last years, little attention has been devoted to analysing the practical benefits and shortcomings of agent technology when applied to such domain. In particular, it is necessary to assess the computational cost added by agent technology in this type of applications so that we can diagnose the improvements required by state-of-the-art agent technology.

Research method:
To assess the computation cost and required memory by using agent technology researchers selected an ecommerce agent base application which the agents are able to negotiate during buying. In this way they used an open agent platform as an agent aware negotiation service and the aim was to study the computational cost of agent awareness for the negotiation services so that its users are aware of the type of negotiation scenarios that can acceptably handle when buying and providing agents are involved. In this way, some artificial negotiation scenarios have been generated for testing purpose and different stages considered through evaluation process. Then the required cost and memory during executions are measured and analyzed.

Contribution:
The contribution of this paper is that it assesses the computational cost added by agent technology in ecommerce applications in order to diagnose the improvements required by the state of the art agent technology. According to implementing the case study and analyzing the results they achieved, authors conclude that while agent technology adds a higher level of abstraction and eases inter-platform communication, state-of-the-art agent technologies require further improvements to tackle real-world domains.

Summary of the method:
To implement the case study, four different agent types have been defined; Logger, Manager, Translator and solver. The interaction protocol between these agents has been modeled. This interaction protocol is composed of several interleaved interaction protocols between Buyer and Manager, Manager and Provider, Manager and Translator.
Some artificial negotiation scenarios for testing have been generated. The evaluation starts from the moment at which all the required data are available to the manager agent. By using some checkpoints the process has been partitioned into several stages and the time and memory has been portioned at the beginning and at the end of these stages.
We have analysed the performance of iBundler through a large variety of negotiation scenarios artificially generated by differently setting the parameters such as number
of providers, the number of bids per provider, the number of RFQ items, and the number of items per bid.
The artificial negotiation scenarios we have generated and tested result from all the possible combinations of the following values:
- Number of providers: 25, 50, 75, 100
- Number of bids per provider: 5, 10, 15, 20
- Number of RFQ items: 5, 10, 15, 20
- Number of items per bid: 5, 10, 25, 50

Time performance and memory usage during the following types of negotiation scenarios are measured through 12 different stages of the iBundler’s solving process:
- The number of items contained in a single bid varies (the 5, 10, 25, and 50).
- The number of bids each provider sends varies (the 5, 10, 15, and 20).
- The number of providers varies (the 25, 50, 75, and 100).

Some observations follow from analyzing the results are:
1. The agent-awareness of iBundler is costly. We observe that the percentage of total time employed to solve the winner determination problem is small with respect to agent related tasks.
2. Using the solver component we can easily solve problems of more than 2000 bids in less than one minute, whereas the agent service can handle in reasonable time less than 750 bids.
3. Therefore, small and medium-size negotiation scenarios can be soundly tackled with iBundler. Nonetheless, time performance significantly impoverishes when handling large-size negotiation scenarios.

According to the result of analyzing memory usage, they also conclude that the memory consumption is highly dependent on the ontology structure.

**Methodology:**
This research hasn't concentrated on a specific methodology.

**Case Study:**
The case study has been used in this paper is iBundler which is in the domain of ecommerce. iBundler is an agent-aware decision support service that acting as a combinatorial negotiation solver (solving the winner determination problem) for both multi-item, multi-unit negotiations and auctions. Thus, the service can be employed by both buying agents and auctioneers in combinatorial negotiations and combinatorial reverse auctions respectively.

**Tools:**
For implementing the case study they have used JADE and ILOG CPLEX.

**Results:**
In this paper, the following results have been reported:
- Time measures when varying the number of bids per provider
- Average times spent at the different evaluation stages
- Memory consumption
- Time performance for negotiation scenarios on the edge of acceptability

They also analyzed the results and conclude that offering iBundler as an agent service implies a significant time overload, while the memory use is only slightly affected.
The main cause of such an overload is related to the encoding and the decoding of ontological objects and messages. The message serializations and deserializations, along with ontology checking heavily overload the system as the dimensions of the negotiation scenario grow. Thus, a suitable work-around is to use, at ontology design time, a more synthetic bidding language, in which bids can be expressed more concisely. In addition, it would be also helpful to improve the performances of the JADE modules devoted to the ontology checking and serialization processes.

Further Work:
In this work the authors focused on the service performances in managing big size negotiation scenarios, not on multiple concurrent accesses to the service. They leave such issue as a possible future development. As future work they propose a comparison of iBundler with other distributed solutions such as CORBA or JAVA RMI.

7.9. Allocating Goals to Agent Roles during MAS Requirements Engineering

Jureta et al. in [JUR2005] believe that allocation of goal responsibilities to agent roles in multi-Agent Systems (MAS) influence the degree to which these systems satisfy nonfunctional requirements.

Problem:
There is widespread agreement that nonfunctional requirements need to be considered early in any Software development process in order to assist reasoning about alternative system structures. While various approaches have been proposed to transform nonfunctional requirements into functional system characteristics during system development the specific issue of using nonfunctional requirements to allocate goals to agent roles during the RE step of MAS development has received limited attention, and no systematic approach has been proposed.

Research method:
The goal-to-role allocation approach presented in this paper introduces an additional technique to generate and choose between alternative agent roles in Tropos Methodology, while relying on a formal nonfunctional requirements representation. It can be added to current goal oriented requirements engineering for agent based systems to introduce the role concept in their process and account for the possibility of alternative agent role definitions, allowing encapsulation and modularity nonfunctional requirements to be addressed more adequately. According to these facts, the technique presented in this paper extends the current works to add role concept to current requirements.

Contribution:
The proposed goal allocation approach advances the state of the art in three ways:
- A systematic approach that starts with the identification of nonfunctional requirements and progressively moves towards the generation of and selection between alternative MAS role structures is proposed. It allows the choice of
goal allocation to agent roles to be justified in relation to the identified nonfunctional requirements.
- A novel type of dependency relationship between goals is used to support the generation of, and selection between, alternative MAS role structures.
- Heuristics for generating and selecting alternative goal-to-role allocations are proposed.

Summary of the method:
The proposed goal to role allocation approach consists of three steps:
- Create a consistent goal tree containing precise requirements
- Identify goal dependencies
- Generate and Select between alternative goal-to-role allocations

For creating goal trees, well know discovery techniques to identify a set of nonfunctional and functional requirements are uses and they are modeled as goals. To illustrate the requirements, KAOS framework was used.
To define dependencies between goals it uses refinement links which contains AND-refinements and OR- refinements and nonfunctional as a weaker versions of refinement links relate nonfunctional goals and functional goals. It also introduces a new type of inter-goal relationship which named goal-dependency. By using this notation, the dependency between the goals is identified according to some techniques:
- (I1) If there is at least one MAS property, constrained in predicates that occur in formal specifications of two goals g1 and g2, then there is a goal-dependency between them. The direction of this goal-dependency is undetermined.
- (I2) If temporal operators in formal specifications of goals in a goal-dependency make it possible to establish the sequence of achievement of one in relation to the other goal, then the goal-dependency relationship between them is directed from the goal whose achievement precedes the other goal’s achievement.
- (I3) If the goal-dependency direction cannot be determined using (I2), then domain/solution knowledge can be used to make an assumption and choose the goal in the goal-dependency whose achievement precedes that of the other goal.

The proposed goal-dependency identification approach has some desirable characteristics:
- Undirected goal-dependencies can be found automatically between all goals in the goal tree, as the goals’ formal specifications contain all the necessary information.
- The second step, (I2) may indicate the need for rewriting goal specifications in order to make them more precise.

It also relates individual goal dependencies to MAS nonfunctional requirements. The aim is to know the type of vulnerability that generated by each goal dependency. In this way, it uses an enriched specification with information about two quality variables that measure the degree of goal achievement.

Methodology:
In related works of the paper, different agent-oriented methodologies have been mentioned such as Gaia, Tropos, MaSE. The authors claim that their approach complements the techniques introduced in related work especially those related to
Tropos methodology. They also found their examples on the Gaia methodology schema. But not specific methodology has been focused on the paper.

**Case Study:**
The Tarzan system is a decentralized, distributed Chaumian mix system that enables client applications to seamlessly direct traffic through an anonymous network at the transport layer. Unlike a centralized server-based anonymous traffic system like Anonymizer Tarzan’s P2P nature ensures that no one actor needs to be trusted for anonymous communication at the IP level. Users can direct their traffic into a tunnel through the network of peers, so the packets exiting the tunnel cannot be traced back to the original sender, even in the face of substantial network-wide traffic analysis.

**Tools:**
No specific tool has been used in this paper.

**Further Work:**
Two additional parameters for organizational design discussed in organizational sciences are the allocation of decision rights and the grouping of work in subunits of an organization. Further work is needed to study the tools and methods for integrating these factors in the process of designing MAS organizations.
The proposed qualitative reasoning technique can be extended to integrate quantitative data. The use of goal-dependencies in the analysis of the timed operation of MAS during the RE step will also be addressed.

**8. Conclusion**
During classifying the reviewed papers, we had in mind different aspects of agent and multi-agent systems requirements; Type of requirements, important roles for defining the requirements, requirements engineering phases, methods and models for specifying agent requirements.
To evaluate the current state of the art in agent-oriented requirements engineering, we summarize our survey in the following tables.
Table 3 shows comparison between the agent specific characteristics according to different definitions of agents [AYA2000]) and current researches have been done to specify them.

**Table 3. functional requirements**

<table>
<thead>
<tr>
<th>Functional Requirement</th>
<th>Autonomy</th>
<th>situatedness</th>
<th>Reactiveness</th>
<th>Proactiveness</th>
<th>Learning</th>
<th>Persistency</th>
<th>Sociability</th>
<th>Goal orientation</th>
<th>Reasoning</th>
<th>Adaptivity</th>
<th>Mobility</th>
<th>Benevolence</th>
<th>Delegacy</th>
<th>Competency</th>
<th>Amenability</th>
<th>Discourse</th>
<th>Rationality</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Research</td>
<td>[Weiss]</td>
<td>[Giorgini]</td>
<td>[Boss]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

35
We also summarized the qualitative requirements of agent-oriented software systems and related researches in Table 4.

### Table 4. Non functional requirements

<table>
<thead>
<tr>
<th>Non-Functional Requirement</th>
<th>Functionality</th>
<th>Reliability</th>
<th>Usability</th>
<th>Efficiency</th>
<th>Maintainability</th>
<th>Reusability</th>
<th>Portability</th>
<th>Design Constraints</th>
<th>Implementation Requirements</th>
<th>Interface Requirements</th>
<th>Physical Requires</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Research</td>
<td>[Guessoume]</td>
<td>[Brinn]</td>
<td>[Giovanichi]</td>
<td>[Bruni]</td>
<td>[Giovanichi]</td>
<td>[Dehlinger]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5 shows a summary of the aspects of agent based systems which agent-oriented methodologies consider according to our survey.

In this table, ✓ shows that the methodology (or its extensions) supports that characteristics or aspect of multi-agent system in requirement or analysis phase. It is necessary to mention that we didn’t consider methodology concerns on design or later phases of the methodology. The empty cell shows that we have not find any evidence that methodology concerns that characteristic.

### Table 5. Agent based systems requirements and methodologies concerns

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Agent Environment</th>
<th>Interactions</th>
<th>Information</th>
<th>Computation</th>
<th>Engineering</th>
<th>Knowledge</th>
<th>Resource</th>
<th>Autonomy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAGMA</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASSIVE</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAS</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common KADS</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODAC</td>
<td></td>
<td></td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cossentino</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaia</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tropos</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaSE</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
By reviewing these tables we can conclude that:

- Intelligence as an important factor of agent based systems has not been considered in the methodologies or independent researches. However some methodologies take into account knowledge (such as ODAC and [COS2002][COS2004]) and experience (such as MAS-commonKAD) and some attempts have been done to specify and elicit reasoning and autonomy as characteristic of intelligent systems but there is no attempt to considering intelligence as an crucial requirement of agent based systems.
- Users as an important role at requirements engineering has not been considered in agent based methodologies explicitly. According to table 5, MAGMA and Tropos are methodologies which take into account users as important stakeholders in their methodologies.
- Specification of agent based systems is the activity which takes most attention in methodologies and independent researches of agent based systems. But other activities such as elicitation, validation and verification of requirements and change management of agent based system requirements have not been taken enough attention. Since we did not consider modeling researches in this survey we do not consider analysis and modeling activity in our conclusion. However we think that because of vast amount of research on modeling of agent based system, this activity is in a good state.

9. Further work
In this section, we specify the future works to continue our research on this field and extending the results of this survey.

1. Since requirements engineering has different activities which relates it to all phases of software development, there is a high cohesion between requirement and other disciplines of software development. Because of that reviewing the research papers of other section of the agent oriented software engineering branch of our research tree (Especially as Analysis and Design, Testing, Methodologies and Modeling) helps us to extend and improve the quality of this research.

2. This survey contains the papers up to 2007 and it does not enclose the papers of the year 2008. Adding the recent researches on agent bases systems helps us to find the latest state of the art in this field.

3. In addition to the surveys we used in this paper, there is another survey entitled “A Survey on agent-oriented software engineering researches”. Because we do not access to this paper at the moment, this survey has not included to this technical report. Adding this survey enriches the Section 6 of this report.

4. It is obvious that creating technical report on other branches of our research tree is possible by following our research approach. Because the papers are introduced and the approach is established, reporting the state of the art for other sections is paved.
5. Categorizing and classifying papers of the Application branch of our research tree helps us to find the domains that take the most benefit from agent technology by applying it.
10. Resources
In this section we present the title of the papers we used in our survey. We also mention the conference or Journals it belongs to and the year the paper published.

10.1. Agents and Multi-agent Systems

10.1.1. Searching

1. Searching for close alternative Plans (AAMAS 2007)

2. Speeding up Moving-Target Search (AAMAS Conf. 2007)

3. Solving Large TÆMS Problems Efficiently by Selective Exploration and Decomposition (AAMAS Conf. 2007)


5. Sequential Decision Making in Parallel Two-Sided Economic Search (AAMAS Conf. 2007)

6. Automatic Feature Extraction for Autonomous General Game Playing Agents (AAMAS Conf. 2007)


8. Support-based Distributed Search (AAMAS Conf. 2006)

9. Integrating Parallel Interactions into Cooperative Search (AAMAS Conf. 2006)


10.1.2. Reasoning

1. Towards Using Multiple Cues for Robust Object Recognition (AAMAS Conf. 2007)

2. Reasoning about Judgment Aggregation (AAMAS Conf. 2007)

3. Reasoning about Action and Cooperation (AAMAS Conf. 2007)
4. Preprocessing Techniques for Accelerating the DCOP Algorithm ADOPT (AAMAS Conf. 2005)

5. A Logic for Strategic Reasoning (AAMAS Conf. 2005)

10.1.3. Learning

2. Learning to Communicate in a decentralized environment (AAMAS 2007)

3. Local Strategy learning in networked multi-agent Team Formation (AAMAS 2007)

4. Reaching Pareto-Optimality in prisoners dilemma using conditional Joint action learning (AAMAS 2007)

5. Monitoring group behavior in goal-directed agents using co-efficient plan observation (AOSE 2006)

10.1.4. Environment
1. Applications and Environments in multi-agent systems (AAMAS 2007)

2. Environments as a first class abstraction in multi-agent systems (AAMAS 2007)

3. Infrastructures for the environment of multi-agent systems (AAMAS 2007)


5. Modeling dynamic environments in multi-agent simulation (AAMAS 2007)


10.1.5. Coordination and collaboration

1. Normative Spaces in Institutional Environments by the means of Commitments, Reputation and Colored Petri Nets (AOSE 2007)

2. Coordinating Microscopic robots in viscous fluids (AAMAS 2007)

3. Coach Planning with opponent models for distributed execution (AAMAS 2006)


5. Distributed Norm Management in Regulated multi-Agent Systems (AAMAS Conf. 2007)

6. Effective Tag Mechanisms for Evolving Coordination (AAMAS Conf. 2007)

7. Matrix-Based Representation for Coordination Fault Detection-A Formal Approach (AAMAS Conf. 2007)

8. Eliciting Single-Peaked Preferences Using Comparison Queries (AAMAS Conf. 2007)


11. Distributed Management of Flexible Times Schedules (AAMAS Conf. 2007)


13. A Globally Optimal Online Algorithm for TTD-MDPs (AAMAS Conf. 2007)

14. A Utility based sensing and communication model for a glacial sensor network (AAMAS Conf. 2006)

15. Accident or Intention- That Is the Question (in the Noisy Iterated Prisoner's Dilemma) (AAMAS Conf. 2006)


17. Agent Interaction in Distributed MDPs and its Implications on Complexity (AAMAS Conf. 2006)

18. Analyzing characteristics of task structures to develop GPGP coordination mechanisms (AAMAS Conf. 2006)
19. Examining DCSP Coordination Tradeoffs (AAMAS Conf. 2006)

20. Social Coordination without Communication in multi-Agent Territory Exploration Tasks (AAMAS Conf. 2006)


22. Knowledge and Social Laws (AAMAS Conf. 2005)


25. Progressive Ontology Alignment for Meaning Coordination An Information-Theoretic Foundation (AAMAS Conf. 2005)

26. Programming Stigmergic Coordination with the TOTA Middleware (AAMAS Conf. 2005)

27. An Integrated Token-Based Algorithm for Scalable Coordination (AAMAS Conf. 2005)


30. Coordinating Multiple Rovers with Interdependent Science Objectives (AAMAS Conf. 2005)

31. Diagnosing a Team of Agents Scaling-Up (AAMAS Conf. 2005)

32. Building intelligent collaborative interface agents with the ICAGENT development framework (AAMAS 2006)

33. Hierarchical multi-agent reinforcement learning (AAMAS 2006)

34. Towards Collaborative Task and Team Maintenance (AAMAS Conf. 2007)

35. Policy Recognition for multi-Player Tactical Scenarios (AAMAS Conf. 2007)

36. Children in the forest- towards a canonical problem of spatio-temporal collaboration (AAMAS Conf. 2007)

37. Coalition Formation under Uncertainty- Bargaining Equilibrium and the Bayesian Core Stability Concept (AAMAS Conf. 2007)


40. Forming Efficient Agent Groups for Completing Complex Tasks (AAMAS Conf. 2006)


42. Predicting Agent Strategy Mix of Evolving Populations (AAMAS Conf. 2005)

43. Task Inference and Distributed Task Management in the Centibots Robotic System (AAMAS Conf. 2005)

44. Task Delegation using Experience-Based multi-Dimensional Trust (AAMAS Conf. 2005)

45. On the Dynamics of Delegation, Cooperation, and Control- A Logical Account (AAMAS Conf. 2005)


47. Extending the Recognition-Primed Decision Model to Support Human-Agent Collaboration (AAMAS Conf. 2005)

48. Effect of referrals on convergence to satisficing distributions (AAMAS Conf. 2005)


10.1.6. Planning

1. Q-value functions for decentralized POMDPs (AAMAS Conf. 2007)

2. On Opportunistic Techniques for Solving Decentralized MDPs with Temporal Constraints (AAMAS Conf. 2007)

3. Collaboration among a satellite swarm (AAMAS Conf. 2007)

4. Aborting Goals and Plans in BDI Agents (AAMAS Conf. 2007)
5. Letting loose a SPIDER on a network of POMDPs- Generating quality guaranteed policies (AAMAS Conf. 2007)

6. Distributed Path Planning for Mobile Robots using a Swarm of Interacting Reinforcement Learners (AAMAS Conf. 2007)

7. Graphical Models for Online Solutions to Interactive POMDPs (AAMAS Conf. 2007)

8. Dynamics Based Control with an Application to Area-Sweeping Problems (AAMAS Conf. 2007)

9. Decentralized planning under uncertainty for teams of communicating agents (AAMAS Conf. 2006)


11. Winning back the cup for distributed POMDPs- Planning over continuous belief spaces (AAMAS Conf. 2006)


17. A Distributed Framework for Solving the multi-agent Plan Coordination Problem (AAMAS Conf. 2005)

10.1.7. Organizational structure

1. Organizational Self-Design in Semi-dynamic Environments (AAMAS Conf. 2007)


3. Agent-Organized Networks for Dynamic Team Formation (AAMAS Conf. 2005)

5. How to Model Inter-Organisational Networks to Enable Dynamic Analyses via Simulations (AOIS 2007)

10.1.8. Ontology

1. A multi-Agent System for Building Dynamic Ontologies (AAMAS Conf. 2007)


3. ANEMONE An Effective Minimal Ontology Negotiation Environment (AAMAS Conf. 2006)


10.1.9. Cognitive Concepts

1. Realistic Cognitive Load Modeling for Enhancing Shared Mental Models in Human-Agent Collaboration (AAMAS Conf. 2007)

2. Interactions between market barriers and communication networks in marketing systems (AAMAS Conf. 2007)

3. Cognitive and Social Simulation of Criminal Behaviour (AAMAS Conf. 2007)


5. Incentive Compatible Ranking Systems (AAMAS Conf. 2007)

6. Normative System games (AAMAS Conf. 2007)

7. Agents, Beliefs, and Plausible Behavior in a Temporal Setting (AAMAS Conf. 2007)

8. Factoring Games to Isolate Strategic Interactions (AAMAS Conf. 2007)


10. Mechanism Design and Deliberative Agents (AAMAS Conf. 2005)
11. ALMA - Layered Model of Affect (AAMAS Conf. 2005)


10.1.10. Social Aspects

1. An integrated trust and reputation model for open multi-agent systems (AAMAS 2006)


4. TRAVOS: Trust and reputation in the context of inaccurate information sources (AAMAS 2006)

5. A Secure Modular Mobile Agent System (SELMAS 2006)

6. Reasoning about Willingness in Networks of Agents (SELMAS 2006)

7. Rumors and Reputation: Evaluating multi-Dimensional Trust within a Decentralized Reputation System (AAMAS Conf. 2007)

8. Resolving Conflict and Inconsistency in Norm-Regulated Virtual Organizations (AAMAS Conf. 2007)

9. Determining Confidence When Integrating Contributions from Multiple Agents (AAMAS Conf. 2007)


11. Using CHI-Scores to Reward Honest Feedback from Repeated Interactions (AAMAS Conf. 2006)

13. An Information-Based model for Trust (AAMAS Conf. 2005)


15. Trusted Kernel-Based Coalition Formation (AAMAS Conf. 2005)


17. Catch Me If You Can -- Exploring Lying Agents in Social Settings (AAMAS Conf. 2005)

18. The Value of Privacy (AAMAS Conf. 2005)

19. Coping with Inaccurate Reputation Sources- Experimental Analysis of a Probabilistic Trust Model (AAMAS Conf. 2005)

20. Enforceable social laws (AAMAS Conf. 2005)

10.1.11. Communication

1. Agent communication and artificial institutions (AAMAS 2007)

2. An Algebra for commitment protocols (AAMAS 2007)

3. Conversational Semantics sustained by commitments (AAMAS 2007)

4. Modeling conversation policies using permissions and obligations (AAMAS 2007)


6. An Approach for Contextual Regulations in Open MAS (AOIS 2006)

7. DIAGAL: An agent communication language based on dialogue games and sustained by social commitments (AAMAS 2006)

8. Handling communication restrictions and team formation in congestion games (AAMAS 2006)


10. Opportunistic Exploitation of Knowledge to Increase Predictability of Agent Interactions in MANETs (SELMAS 2005)

11. Evaluating a Conversation-Centered Interactive drama (AAMAS Conf. 2007)
12. Dynamic Semantics for Agent Communication Languages (AAMAS Conf. 2007)

13. Commitment and Extortion (AAMAS Conf. 2007)


15. Enacting Protocols by Commitment Concession (AAMAS Conf. 2007)

16. Implementing Commitment-Based Interactions (AAMAS Conf. 2007)

17. Communication Management Using Abstraction in Distributed Bayesian Networks (AAMAS Conf. 2006)

18. Specifying and Resolving Preferences Among Agent Interaction Patterns (AAMAS Conf. 2006)


23. Formalizing and Achieving Multiparty Agreements via Commitments (AAMAS Conf. 2005)


26. Dynamic Epistemic Logic with Assignment (AAMAS Conf. 2005)


**10.1.12. Negotiation**

1. Ontology negotiation: goals, requirements and implementation (IJAOSE 2007)

2. A negotiation meta strategy combining trade-off and concession moves (AAMAS 2006)

3. Ontologies for supporting negotiation in e-commerce (EAofAI 2005)

5. On the Relevance of Utterances in Formal Inter-agent Dialogues (AAMAS Conf. 2007)

6. Negotiation by abduction and relaxation (AAMAS Conf. 2007)


8. Arguing and explaining classifications (AAMAS Conf. 2007)


11. Argumentation over Ontology Correspondences in MAS (AAMAS Conf. 2007)


13. Approximate and online multi-issue negotiation (AAMAS Conf. 2007)

14. Tractable Negotiation in Tree-structured Domains (AAMAS Conf. 2006)

15. An Argumentation-based Approach for Practical Reasoning (AAMAS Conf. 2006)


17. Monotonic Concession Protocols for Multilateral Negotiation (AAMAS Conf. 2006)

18. Negotiating using Rewards (AAMAS Conf. 2006)

19. An Automated Negotiation Technique for Self-interest Agents (CSICC 2007)


21. Adapting to Agents' Personalities in Negotiation (AAMAS Conf. 2005)

22. What Kind of Argument Are We Going to Have Today (AAMAS Conf. 2005)

23. Argumentation-based dialogues for deliberation (AAMAS Conf. 2005)

25. Negotiating over Small Bundles of Resources (AAMAS Conf. 2005)


10.2. Agent-oriented Software Engineering

10.2.1. Requirements

1. Goal-oriented requirements analysis and reasoning in the Tropos methodology (EAofAI 2005)


4. Operational Modeling of Agent Autonomy (AOSE 2005)

5. Requirements Analysis of an Agent’s Reasoning Capability (AOIS 2005)

6. Incorporating Commitment Protocol into Tropos (AOSE 2005)

7. On the Cost of Agent-awareness for Negotiation Services (AOIS 2005)

8. Contributions of Quantitative Modeling to Agent-Oriented Software Engineering

9. Allocating Goals to Agent Roles during MAS Requirements Engineering

10.2.2. Analysis and Design

1. Modeling the Provenance of Data in Autonomous Systems (AAMAS Conf. 2007)

2. Contextualizing Commitment Protocols (AAMAS Conf. 2007)

3. A Process for Analyzing Agent-Oriented Patterns (AOSE 2007)

5. Integrating MAS in a Component-Based Groupware Environment (AOSE 2006)


8. Views: Customizable Abstractions for ContextAware Applications in MANETs (SELMAS 2005)


11. Hermes- Designing Goal Oriented Agent Interactions (AOSE 2005)

12. Formalization and Analysis of the Temporal Dynamics of Conditioning (AOSE 2005)


16. Multi-agent architectures as organizational structures (AAMAS 2006)


18. A multiagent system for the reliable execution of automatically composed ad-hoc processes (AAMAS2006)


20. From SMART to agent systems development (EAofAI 2005)


22. A Secure Modular Mobile Agent System (SELMAS 2006)


27. Building Reliable Systems based on Self Organizing MultiAgent Systems (SELMAS 2006)

28. Improving the Architectural Design of Multi-Agent Systems: The Tropos Case (SELMAS 2006) Also Methodology


32. Goals in the Context of BDI Plan Failure and Planning (AAMAS Conf. 2007) (Also Formal Modeling)

33. Modular BDI Architecture (AAMAS Conf. 2007)

34. U-DIRECTOR- A Decision-Theoretic Narrative Planning Architecture for Storytelling Environments (AAMAS Conf. 2006) (Also Planning)


36. An Agent-Based Meta-Level Architecture for Strategic Reasoning in Naval Planning (AOIS 2005)

37. INCA (Investor Network Collaborative Architecture) - A Method in the Madness of Wall Street (AOIS 2005)

38. A Hybrid Three Layer Architecture for Fire Agent Management in Rescue Simulation Environment (CSICC 2006)


10.2.3. Implementation

1. An Expressway from Agent-Oriented Models to Prototype Systems (AOSE 2007)


3. Engineering open environments with electronic institutions (EAofAI 2005)
4. Open Agent Systems (AOSE 2007)

5. Applying ForMAAD for Designing the Air Traffic Control Application (AOSE 2006)


8. Self-Organizing Social and Spatial Networks under What-if Scenarios (AAMAS Conf. 2007)

9. Bidding Algorithms for a Distributed Combinatorial Auction (AAMAS Conf. 2007)

10. Goal-Oriented Modularity in Agent Programming (AAMAS Conf. 2006)

10.2.4. Testing
1. A Goal-Oriented Software Testing Methodology (AOSE 2007)

2. Systematic verification of multi-agent systems based on rigorous executable specifications (IJAOSE 2007)

3. Model integration in agent-oriented development (IJAOSE 2007)


5. Verifying multi-agent programs by model checking (AAMAS 2006)

6. Adding debugging support to the Prometheus methodology (EAofAI 2005)


11. An Aspect-Oriented Approach for Modeling Self-Organizing Emergent Structures (SELMAS 2006)

12. Modular Interpreted Systems (AAMAS Conf. 2007)
13. A complete and decidable security-specialised logic and its application to the TESLA protocol (AAMAS Conf. 2006)


15. On the Complexity of Practical ATL Model Checking (AAMAS Conf. 2006)


18. Implementing Validated Agent Behaviors with Automata based on Goal- Decomposition Trees (AOSE 2005)


10.2.5. Methodology

1. A Methodology for developing MAS as 3D Electronic Institutions (AOSE 2007)


3. Extending Gaia with Agent Design and Iterative Development (AOSE 2007)

4. AgentPrIMe: Adapting MAS Designs to Build Confidence (AOSE 2007)

5. Refining Goal Models by Evaluating System Behaviour (AOSE 2007)

6. Method fragments for agent design methodologies: from standardisation to research (IJAOSE 2007)


11. Estimation of Complexity in Agent Oriented Methodologies via Evaluation of Models and Artifacts (CSICC 2007)


15. Thespian- Using Multi-Agent Fitting to Craft Interactive Drama (AAMAS Conf. 2005)

16. Identification of Reusable Method Fragments from the PASSI Agent-Oriented Methodology (AOIS 2005)


10.2.6. Modeling


2. Modeling Mental States in the Analysis of Multiagent Systems Requirements (AOSE 2006)

3. Extending UML Sequence Diagrams to Model Agent Mobility (AOSE 2006)

4. Autonomous agent modeling using I/O Automata (AOSE 2006)


6. Towards Agent-Based Scenario Development for Strategic Decision Support (AOIS 2006)

7. UML 2.0 and agents: how to build agent-based systems with the new UML standard (EAofAI 2005)


10. An Ontology Based Multi-Agent System Conceptual Model (AOSD 2006)

11. An Adversarial Environment Model for Bounded Rational Agents in Zero-Sum Interactions (AAMAS Conf. 2007)


14. Contributions of Quantitative Modeling to AOSE (AOSE 2005)

15. Simplifying i* Models (AOIS 2007)

16. The Meaning of Inheritance in i*  (AOIS 2007)

17. Supporting the Development of Multi-Agent Interactions via Roles (AOSE 2005)

10.3. Agent-oriented Applications

1. Constraint Satisfaction Algorithms For Graphical Games (AAMAS Conf. 2007)

2. A Randomized Method For The Shapley Value For The Voting Game (AAMAS Conf. 2007)


4. A Computational Characterization Of Multi-Agent Games With Fallacious Rewards (AAMAS Conf. 2007)

5. Advanced Bidding Agent For Advertisement Selection On Public Displays (AAMAS Conf. 2007)

6. An Efficient Heuristic Approach For Security Against Multiple Adversaries (AAMAS Conf. 2007)

7. Better Automated Abstraction Techniques For Imperfect Information Games, With Application To Texas Hold'em Poker (AAMAS Conf. 2007)


10. A Technique For Reducing Normal Form Games To Compute A Nash Equilibrium (AAMAS Conf. 2006)


12. Camera-Based Observation Of Football Games For Analyzing Multi-Agent Activities (AAMAS Conf. 2006)


14. Designing A Successful Trading Agent For Supply Chain Management (AAMAS Conf. 2006)
16. Efficient Agent-Based Cluster Ensembles (AAMAS Conf. 2006)
17. Embodied Mobile Agents (AAMAS Conf. 2006)
18. Power And Negotiation - Lessons From Agent-Based Participatory Simulations (AAMAS Conf. 2006)
19. Robust Game Play Against Unknown Opponents (AAMAS Conf. 2006)
20. Resource Selection Games With Unknown Number Of Players (AAMAS Conf. 2006)
23. RPD-Enabled Agents Teaming With Human For Multi-Context Decision Making (AAMAS Conf. 2006)
24. False-Name-Proof Combinatorial Auction Protocol (AAMAS Conf. 2006)
25. On The Logic Of Coalitional Games (AAMAS Conf. 2006)
26. Solution Sets For Dcops And Graphical Games (AAMAS Conf. 2006)
27. Multi-Model Motion Tracking Under Multi-Agent Actuators (AAMAS Conf. 2006)
29. Temporal Qualitative Coalitional Games (AAMAS Conf. 2006)
30. The Use Of Agents In Human Learning System (AAMAS Conf. 2006)
31. Mertacor-A Successful Autonomous Trading Agent (AAMAS Conf. 2006)
32. Reagents- Behavior-Based Remote Agents And Their Performance (AAMAS Conf. 2006)
33. Performance Of Pheromone Model For Predicting Traffic Congestion (AAMAS Conf. 2006)
34. Games With Possibly Unaware Players (AAMAS Conf. 2006)
35. Instantiating The Contingent Bids Model Of Truthful Interdependent Value (AAMAS Conf. 2006)
36. Scalable And Reliable Data Delivery In Mobile Ad Hoc Sensor Networks (AAMAS Conf. 2006)

37. Monopolising Markets By Exploiting Trust. (AAMAS Conf. 2006)

38. Failures Of The VCG Mechanism In Combinatorial Auctions And Exchanges (AAMAS Conf. 2006)

39. Adapted Information Retrieval In Web Information Systems Using PUMAS (AOIS 2005)

40. A Cooperative Multi-Agent Approach To Free Flight (AAMAS Conf. 2005)

41. Toward An Understanding Of The Impact Of Personal Assistants On Human Organizations (AAMAS Conf. 2006)

42. Gradient Field-Based Task Assignment In An AGV Transportation System. (AAMAS Conf. 2006)

43. Evaluation of a Mobile Agent Based Intrusion Detection System (CSICC2007) (In Persian)

44. Secure Key Management in Hierarchical Group Communication Using Agents (CSICC 2007) (In Persian)


46. Valuations Of Possible States (VPS) A Quantitative Framework For Analysis Of Privacy Loss Among Collaborative Personal Assistant Agents (AAMAS Conf. 2005)

47. Using Tabu Best-Response Search To Find Pure Strategy Nash Equilibria In Normal Form Games (AAMAS Conf. 2005)


49. Personalisation For User Agents (AAMAS Conf. 2005)


51. Design of Trading Agent with BDI Architecture (CSICC 2007) (In Persian)

52. A Hybrid Method for Protecting the Integrity of Mobile Agents (CSICC 2007) (In Persian)

54. Swarm Intelligence: a Novel Approach for Agent-based Intelligent Information Retrieval (CSICC 2007) (In Persian)

55. GASP: Parallel Skeleton Programming Based on Mobile Agents on Grid (CSICC 2005) (In Persian)

56. Implicit- An Agent-Based Recommendation System For Web Search (AAMAS Conf. 2005)

57. Implementing Norms In Electronic Institutions (AAMAS Conf. 2005)


59. Agent-Based Trust Model Involving Multiple Qualities (AAMAS Conf. 2005)

60. Multi-Agent Computer Network Routing System Based On Recursive Modeling Method And Bayesian Belief Update (CSICC 2007)


63. Towards A Theory Of Local To Global In Distributed Multi-Agent Systems (I) (AAMAS Conf. 2005)

64. Towards A Theory Of Local To Global In Distributed Multi-Agent Systems (II) (AAMAS Conf. 2005)

65. Sequential Auctions For Objects With Common And Private Values (AAMAS Conf. 2005)


67. Comparing Two Approaches To Dynamic, Distributed Constraint Satisfaction (AAMAS Conf. 2005)

68. Computationally-Efficient Combinatorial Auctions For Resource Allocation In Weaklycoupled Mdp's (AAMAS Conf. 2005)

69. Decentralized Voting With Unconditional Privacy (AAMAS Conf. 2005)

70. Multi-Agent Decision Support Via User-Modeling (AAMAS Conf. 2005)

71. Multi-Agent Traffic Management An Improved Control Mechanism. (AAMAS Conf. 2005)
72. Formalization Of A Voting Protocol For Virtual Organizations  (AAMAS Conf. 2005)

73. Exploiting Belief Bounds- Practical Pmdps For Personal Assistant Agents  
(AAMAS Conf. 2005)

74. Exploiting A Sensed Environment To Improve Human-Agent Communication  
(AAMAS Conf. 2005)

75. Emerging Collective Behavior In A Simple Artificial Financial Market  
(AAMAS Conf. 2005)


77. Architecture For Distributed Agent-Basedworkflow (AOIS 2005)

78. Supporting Program Indexing And Querying In Source Code Digital Libraries  
(AOIS 2005)

79. Design Options For Subscription Managers  (AOIS 2005)

80. A Knowledge Management Framework For Semantic Multi-Agent Systems  
(AOIS 2007)

81. A Survey Of Intelligent Agents For User Modeling On The WEB And 
Modeling The User Behavior In Question Answering System  (CSICC 2006)

82. DANA- An Agent with the Capability of Understanding and Executing  
Persian Text (CSICC 2006) (In Persian)

83. An Agent- Based Model for Finding the Optimum in Banking Portfolio  
(CSICC 2006) (In Persian)

84. Using Intelligence Agents with Human Guidance in Computational  
Environments for Automatic Service Selection  (CSICC 2006) (In Persian)

85. A New Approach for Testing Web Applications Using Software Agents  
(CSICC 2005) (In Persian)

86. Applying Formaad For Designing The Air Traffic Control Application (AOSE 2006)

87. Toothagent: A Multi-Agent System For Virtual Communities Support (AOIS 2006)

88. A Content-Based Information Sharing Multi-Agent (AOIS 2006)

89. MLBP: MAS For Largescale Biometric Pattern Recognition (AOIS 2006)
90. Outperforming The Competition In Multi-Unit Sealed Bid Auctions (AAMAS Conf. 2007)

91. On The Winner Determination Problem In Mixed Multi-Unit Combinatorial Auctions (AAMAS Conf. 2007)

92. On The Benefits Of Cheating By Self-Interested Agents In Vehicular Networks (AAMAS Conf. 2007)

93. Distributed Agent-Based Air Traffic Flow Management (AAMAS Conf. 2007)

94. Empirical Game-Theoretic Analysis Of The TAC Supply Chain Game (AAMAS Conf. 2007)

95. Bidding Algorithms For A Distributed Combinatorial Auction (AAMAS Conf. 2007)

96. An Agent-Based Approach For Privacy-Preserving Recommender Systems (AAMAS Conf. 2007)

11. References


International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS’03), Lecture Notes in Computer Science 2691, Springer Verlag, pp. 394-403.


