UNIVERSITÁ DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

SCUOLA DI DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE
INDIRIZZO: INGEGNERIA INFORMATICA ED ELETTRONICA INDUSTRIALI
CICLO XX

CONTEXT-DEPENDENT REPUTATION MANAGEMENT IN MULTI-AGENT SYSTEMS

Direttore della Scuola: **Ch.mo Prof. Matteo Bertocco**
Supervisore: **Ch.mo Prof. Carlo Ferrari**

Dottorando: **Cristian Bertocco**

Sommario

Il paradigma di programmazione orientato agli agenti e gli agenti software stanno ottenendo l'attenzione della comunitá dell'Information Technology in quanto sembrano essere una soluzione promettente per lo sviluppo di sistemi che si possano applicare alle discipline emergenti del *mobile computing*, *ubiquitous computing* e *pervasive computing*.
Gli agenti software consistono in processi autonomi che vengono eseguiti in piattaforme host, ed hanno l'obiettivo di raggiungere gli scopi per i quali i proprietari li hanno programmati. Gli agenti hanno la capacitá di migrare da un host ad un altro sfruttando le risorse e i servizi che sono disponibili nel sistema distribuito.

Negli anni novanta, agli inizi della definizione del modello ad agenti, i sistemi ad agenti erano utilizzati per applicazioni che avevano prospettive strettamente locali: tutto il sistema era contenuto all'interno di frontiere ben definite ed era composto di un definito insieme di agenti e risorse di calcolo tra di loro reciprocamente noti. I sistemi ad agenti stanno ora ampliando il loro utilizzo in applicazioni con prospettive piú ampie: le piattaforme ad agenti vengono utilizzate in sistemi distribuiti caratterizzati da ampie dimensioni e da un'alta dinamicitá senza confini ben definiti.

I problemi di sicurezza che sorgono nei sistemi aperti ad agenti richiedono nuovi meccanismi di protezione che combinino la protezione da agenti maliziosi con le frontiere dei sistemi locali aperte alla vasta comunitá di potenziali utilizzatori. I sistemi di sicurezza definiti *soft* sono basati sull'impiego dei concetti di fiducia e reputazione, i quali sono derivati dalle stesse definizioni usate comunemente nelle relazioni sociali. La sicurezza *soft* é utilizzata nelle decisioni basate sul grado di fiducia: essa é impiegata nella valutazione dei rischi che possono derivare da interazioni con agenti non conosciuti. Le reputazione rendono danno informazione agli agenti sul tipo di comportamento e il grado di onestá di altri agenti nella comunitá, in base alle opinioni che derivano da interazioni avvenute precedentemente.

Viene proposto un modello per la gestione della reputazione che sia coerente con le caratteristiche di un sitema ad agenti (per esempio la dinamicitá degli agenti) e che rispetti anche i requisiti che sorgono in questi sistemi in particolari situazioni critiche, quali per esempio capacitá computazionali ridotte dei dispositivi portatili sui quali vengono eseguiti gli agenti e comunicazione instabile in dispositivi mobili.

Il modello si basa sulla valutazione di opinioni che vengono raccolte all'interno di gruppi di contesto. Un gruppo di contesto é una coalizione di agenti i quali forniscono opinioni riguardanti il comportamento degli agenti. In un gruppo di contesto le opinioni sono legate allo specifico contesto. Il modello proposto quindi introduce la caratteristica di contesto nelle informazioni di reputazione.
Il modello di reputazione fornisce agli agenti informazioni che riguardano sia l'onestá generale di un agente, sia il suo comportamento rispetto il contesto dove esso viene valutato.

Il modello si compone anche di una rete di punti di informazione distribuiti che hanno lo scopo di supportare lo distribuzione delle informazioni fra gruppi di contesto.

Il modello é consistente sia dal punto di vista del modello agente sia da quello della sua dimensione sociale, in quanto é stato disegnato attorno al concetto di agente ma allo stesso tempo sono state adottate soluzioni che permettano il suo utilizzo in ampie societá di agenti.

Abstract

Agent-oriented programming paradigm and mobile software agents are getting the attention of IT community because they look a promising solution for the development of systems that fit the computational requirements of emerging mobile computing, ubiquitous computing and pervasive computing.
Software agents consist of autonomous processes that run on a hosting platform and that are aimed at achieving specific owners' goals they were programmed for. Agents can migrate from host to host in order to exploit resources and services that are available in the distributed system.

In the 90's, at the beginning of the agent model definition and development, agent systems addressed applications with strictly local perspectives: the whole system was contained within a defined boundary and was composed of a defined set of known agents and hosts.
Agent systems are now moving towards open perspective applications: agent platforms are involving in wide size and higly dynamic distributed systems without well-defined boundaries.

Security issues that arise in open agent systems require new protection mechanisms that combine the protection from malicious agents with local boundaries that are open to the community of users.
Soft security mechanisms are based on the employment of trust and reputation concepts that are derived from the same definitions used in social relationships. Soft security is employed in trust-based decision making: it support the evaluation of risks that are involved in interactions with unknown agents; reputation information informs agents about the behavior and honesty of agents in the community according to the opinions coming from experienced interactions.

We propose a model for reputation management that is compliant with the characteristics of the agent systems (e.g. dynamicity of agents), and that meets also requirements that arise in these systems in critical situations, e.g. low computational power in small portable devices that host agents and low and unstable communication bandwidth in mobile devices.

The model is based on the evaluation of opinions that are collected within context groups. A context group is a coalition of agents that provide opinions related to the

behaviors of agents. In a context group opinions are related to a specific context. Hence the proposed model introduces a context feature to reputation information.
The reputation model provides agents with information that regards both the general fairness of an agent and its behavior related to the context where it is rated.

The model is also composed of a network of distributed informative points that are aimed at supporting trust information sharing among context groups.

The model is consistent for what concerns both the agent model and the agent social perspective because its design adopted an agent-centric approach but at the same time it adopted solutions in order to employ the model in large society of agents.

Acknowledgments

First and foremost I would like to thank my supervisor Prof. Carlo Ferrari who guided me during these years. I am very grateful to him for every minute he spent to teach me how to improve my results and for any precious suggestion he gave me.

I want to thank also Matteo, Alberto, Marco, Rudi, Mauro, Cinzia, Natalí, Yuri, Cucciolo, Paolo, Versacif, Fabio, Giovanni, Claudio, Sandra. I really enjoyed the time spent with you at the DEI.

Many thanks to my family and the Montaño family and to all other friends of mine that I did not mention above: my friends from Saccolongo, those I met in Davis, those from Agordo, and from Monselice. Life is much better with you.

Thanks.

Contents

“Dov’é Jones il suonatore
che fu sorpreso dai suoi novant’anni
e con la vita avrebbe ancora giocato.

Lui che offrì la faccia al vento
la gola al vino e mai un pensiero
non al denaro, non all’amore né al cielo.

Sembra di sentirlo ancora
dire al mercante di liquore
"Tu che lo vendi, cosa ti compri di migliore?"”

(F.d.A.)

Chapter 1

Agent Paradigm and Multi-Agent Systems

1.1 Introduction

Rosenschein in the MIT Encyclopedia of Cognitive Science [80] proposes a definition of software agent:

"An intelligent agent is a device that interacts with its environment in flexible, goal-directed ways, recognizing important states of the environment and acting to achieve desired results. Clearly, when designing a particular agent, many domain-specific features of the environment must be reflected in the detailed design of the agent. Still, the general form of the subsystems underlying intelligent interaction with the environment may carry over from domain to domain. Intelligent agent architectures attempt to capture these general forms and to enforce basic system properties such as soundness of reasoning, efficiency of response, or interruptibility. Many architectures have been proposed that emphasize one or another of these properties, and these architectures can be usefully grouped into three broad categories: the deliberative, the reactive, or the distributed."

Agent-based systems are an important area of research and development emerged in information technology in the 1990s. Agents are considered one of the most important new paradigm for software development since object-orientation [57]. Agent software paradigm brings concepts from the theories of artificial intelligence into the mainstream field of distributed systems. Agents are a way to manage interactions between large-scale distributed computational systems. The concept of agent is a repository where both AI and software engineering can merge their efforts, thus providing new tools for future IT development. An application that is developed according to the agent paradigm, is modeled as a collection of components called agents that are essentially characterized by autonomy, proactivity and ability to communicate. These

features are described in more details in this section. A system composed of agents is called Multi-Agent System (MAS).

A software agent is an autonomous process with expertise that is entrusted to go out and act on the owner's behalf, helping him to achieve computing goals.
Autonomy means that it is capable of modifying the way in which they act to achieve their goals, they can make decisions according to their beliefs and their perceived environment. They can decide for themselves what they need to do, hence they are capable of operating in dynamic and open environments interacting with other agents.

Agent-based computing has been implemented in processes such as automated financial markets trading, logistics, and industrial robotics. Now it is moving into commercial sectors as more complex systems with many different components are used by a wider range of organisations. Among those that have successfully implemented agent technologies there are DaimlerChrysler, IBM and the U.S. Ministry of Defence. Many companies sell agent conceptualization allied to agent-related technologies (Eurobios, Agentis Software, Agent Oriented Software Group, Lost Wax, Magenta Technologies, Whitestein Technologies, Living Systems).

An agent is expected to be [32]:

- social and communicative: an agent is able to collaboratively interact with other entities in the system,
- capable of making decisions: an agents is able to take options according to the owner's goals and preferences rather than simply provide advice;
- autonomous: an agent has autonomous execution capabilities, hence it can sense and act without the user's control the whole time and to perform domain oriented reasonong;
- adaptive: an agent is able to learn from experiences in the environment it lives;
- goal-oriented: an agent is a persistent software entity dedicated to a special purpose;
- reactive and proactive: an agent reacts to the events that happen in its environment, for instance an agent could perceive its environments through sensors of devices distributed over the environment and can react upon that environment through the effectors of the devices; being proactive means that the agent can anticipatory behave, it is able to act according to its own initiative; this term is derived from organizational behavior psychology. Proactivity behavior involves acting in advance of a future situation, rather than just reacting, taking control and making things happen rather than just adjusting to a situation or waiting for something to happen;

- mobile: an agent can migrate from an host to another to if the latter is more suitable for its purpose;

- BDI-style modelled: belief-desire-intention software model is closely associated with intelligent goal-oriented agents; the internal state and decision process of agents are modelled following the notion of mental attitudes; goal-oriented agents do not directly receive requests to perform certain actions, but the developer defines more abstract goals providing a certain degree of flexibility on how to achieve the goals. Agents have beliefs stored in a belief base; they have implicit or explicit goal descriptions (states) to be achieved through the execution of plans stored in a plan library;

- flexible: an agents implements late-binding service invocation methods.

Agents can respond to events that happen in their environment with a set of if-then rules to achieve the goal and desire of the designer, but they are also supposed to react to unexpected circumstances. For instance they can incorporate disciplines such as game theory to plan strategies that lead their behaviors.
For instance software agents were employed in NASA government agency's projects: they were used for the capability of reconfiguring themselves in response to changing conditions. The US defense is employing agents because it is interested in survivable systems able to regenerate lost network communication capabilities. The most compelling agent application is the business-to-business (B2B) one. New York Stock Exchange uses program-trading, software agents, for a relevant proportion of stock transactions [61].

Modern business and transformational military systems need dynamic, flexible, adaptive, fully distributed, agile IT to support highly dynamic processes and coordination models with not known parameters a priori. Software is moving away from individual vertically integrated applications to adopt new paradigm for building custom-tuned networks of applications on a peer-to-peer model [38]. Agent systems employ semantically sophisticated interaction protocols as a flexible control layer that binds themselves in an organized structure.

Rather than being the solution to several computing problems, the agent paradigm provides a conceptualization and an architectural model that can be applied in application domains or software engineering models that are characterized by interactions among self-interested and autonomous entities (e.g. ubiquitous and pervasive computing disciplines [57]).

Several efforts are being spent for research and deployment of agent systems. For instance European commission has funded the AgentLink project since 1998 to support agent research and development. AgentLink's aim is to foster research activity in agent technology:

- promoting and raising awareness of agent systems technology in European industrial societies;
- supporting standardization effort for the sake of interoperability;
- facilitating adoption of agent technology in industrial applications;
- promoting excellence in teaching in this area;
- providing discussion groups in order to share experiences and solutions to agent issues.

Improvements on the traditional computing models and paradigms are required to cope with needs for dynamic and open systems, which require interaction among heterogeneous and rapidly changing systems and organizations. Agent paradigm suits these requirements software agents provide a design metaphor and abstraction tools, for structuring application around autonomous and communicative entities. Complex systems can be considered as distinct and independent components, that interact, in a way that the system appear aggregated and organized in a conceptual whole.

Interactions among entities in heterogeneous systems is a prerogative for the agent paradigm. An agent system can be composed of agents that act without interacting with other peers, for instance agents that operate on behalf of human principals and assist them to carry out some tasks that do not require other agents to be involved. However the agent paradigm models a system as a compound of agents, each embedding specific roles, that collaborate to achieve their own goals, or just one common goal (for instance a joint decision-making process where agents collaborate to evaluate different constraints optimizing results of taken decisions, or a simulation systems where agents model components of a real-world domain such as physics, biology, economy, and social science).

Communication plays an important key rule in agent technology. Agents communicate with users, resources, and with each other to cooperate or negotiate. A common language is required to support full agent functionalities. The most famous common languages are KQML and FIPA ACL. The former was developed by US Government's Agency ARPA, the latter from the Foundation for Intelligent Physical Agents. Both only deal with agent-to-agent communication. Much work has to be done for human-computer interface issues if agents are supposed to act on behalf of a human.

The agent model employs two different architectures:

- purely reactive agents that operate in a simple stimulus-response fashion and that do not embed concepts of plans or beliefs; the Jade platform [1] uses this kind of agents;

- deliberate agents that implement the BDI model (beliefs, desires, intentions) and that embed the concept of intelligent agent; for instance the JACK [76] and JADEX [77] platforms implement those kind of agents.

Industrial and commercial applications where agents have been deployed are related to manufacturing (scheduling and controlling operations, controlling robots), process control (monitoring the system, diagnosing faults, power management), telecommunication, air traffic control, traffic and transportation management (for instance planning supply chain distribution), information filtering and gathering, electronic commerce (full automation is still far but increasing amount of trade is being undertaken by agents, for instance on-line store searching for price and availability), business process management, defence, entertainment, health care. Major companies investing in agent system research and development are IBM, Microsoft, Siemens, HP Labs, BTexact, and some start-ups such as Tryllian (Netherlands), Agent Oriented Software LostWax and Magenta (UK), IKV++ and Living Systems(Germany), Whitestein Technology (Switzerland).

1.2 Mobile agents

Mobile agents is the most promising branch of development for agent systems. Mobile agents are programs that roam networked hosts to achieve owners' goals. For instance their aims can be the collection of data from distributed containers, the execution of computation in better performing hosts, the execution of control operation in remote devices). These agents require standards in order to migrate code from one host to another.

An important reason why mobile agents have not yet been taken up by the mainstream, is security threat involved with roaming software. On the one hand agents need to be protected from malicious hosts, on the other hand hosts need to be protected from agents that could threaten the systems. Mobile agents improve end users' connection to network service providers when they experience low bandwidth (mobile phones and portable devices) by offloading application processing to a node with high bandwidth connectivity. They allow a divide-and-conquer approach to complexity of large-scale distributed and decentralized systems. For instance in information retrieving, the agent can migrate towards the data container in order to directly analyze the data in the place where they are contained.

Computation in mobile agents becomes local to the host, avoiding transaction requirements among applications. This results in improved robustness of the system. Deploying agents to remote hosts results in higher flexibility and adaptability for computation. Hence mobile agents are a natural tool for system integration issues. For instance mobile agents provide an easy solution to monitoring and updating operations

in remote systems, being capable of simultaneously and independently visiting several locations, without any requirement of continuous connectivity among the systems.
Agents can transfer drivers and adapters to source systems to dynamically change the configuration of the environment.
Agents are used for telecommunication network management. For instance mobile networks are composed of several units that perform different tasks (mobile devices, base stations, base station controllers, mobile switch centers). An agent's behavior can be easily changed, hence the user can differentiate the agent's functionalities according to the host it is visiting. Agents are spread to configure the above mentioned components of the network depending on the traffic distribution.

Some of the most famous cornerstones of mobile agent platforms are: *Telescript* from General Magic, *Aglets* IBM [48], *Mole* [79], *TACOMA* from the Universities of Tromso and Cornell [78], *D'Agents* from the Dartmouth College [24], *Jade* from Telecom Italia Labs [1].

Heterogeneity is a typical issue in distributed systems. The agent model requires a layer that hides heterogeneity of the system in order for the agents to run on heterogeneous platforms. An agent platform is the *middleware* software layer that provides common services and supports agent management functionalities to the agents running on an host. It provides agents with an interfaces to interact with the underlying operating system.

1.3 Agent-oriented and Object-oriented programming

A programming paradigm is a way of thinking to the components of systems that brings useful perspective for their development. The agent-oriented programming paradigm (AO) can be considered the effect of the evolution of the object-oriented programming paradigm. Agents are different from objects, but they share many things in common. For instance, agents are build upon objects, but they introduce different features. Agent-based way of thinking is similar to the object-based: for the former everything could be seen as a compound of agents, for the latter everything is made up of objects.

In the OO perspective, objects are the unit components that compose a system. An object can be anything ranging from a concrete entity from the real world to a conceptual entity that only exists in the model. Each object within the system is associated with a particular class that determines the objects basic properties. Classes can be linked with each other in several ways:

- with inheritance: a class can be a conceptual extension of a common base specification class;
- objects communicate by sending messages to each other; these messages can

be used to request services from the receiving object such as to provide internal information or to change the current state;

- an instance of a class can contain an instance of another class.

These concepts correspond to the AO perspective by replacing class with role, state variable with belief/knowledge and method with message. Thus a role definition describes the agent's capabilities, the data that is needed to produce the desired results and the requests that trigger a particular service.

In the agent-oriented universe there is no single agreed definition of agent. The existing agent theories are based upon the agent concepts coming from sociology, artificial intelligence and robotics. An agent is modeled in terms of mental notions such as beliefs, desires and intentions. An agent is also anything that exhibits autonomy, reactivity, pro-activity, social ability.

High-level interaction among agents is compared with interaction between objects that is expression of simple functional dependencies. Furthermore agents are autonomous and proactive; objects are reactive and are not supposed to take decisions, to plan their actions in order to achieve a goal.

Aspect-oriented programming, active objects, context-dependencies in component-based applications are somehow the proof that object-oriented programming is moving towards new concepts and abstractions very close to the characteristics incorporated by agents.

Table Table 1.1 describes the historic development from machine language to agent-oriented programming [88]:

- a program was a monolithic block without any inherent structure;
- a program is made up from several smaller structural units, the subroutines; lack of program control flow and data organization aspects;
- data and computation grouped together in a single structural unit called an object;
- moving from merely passive objects towards active structural units called agents.

Autonomous agents lead to a new way of software system development. Software systems are no longer a compound of passive objects, but they are modeled using active entities. With this approach the designer sets out the initial state and goals of the autonomous agent and the ongoing interactions among agents determine the overall system behavior.

Some examples where the agent-oriented approach could better fit rather than object-oriented approach:

- control systems for physical domains that require autonomous proactive processes,

	MACHINE LANG.	STRUCTURED PROGR.	OBJECT ORIENTED	AGENT ORIENTED
STRUCTURAL UNIT	`program`	`subroutine`	`object`	`agent`
RELATION TO PREVIOUS LEVEL		`bound unit of program`	`subroutine + persistent local state`	`object + independent thread of control + autonomy`

Table 1.1: History of programming evolution.

- integration of distributed applications in mobile devices,
- internet-based distributed computing that requires autonomous and cooperative software with decentralized management,
- mobile and pervasive computing that require to explicitly take into account environmental features and their unpredictable dynamism.

1.4 Application and research domains for agent systems

The following list describes major application areas for the agent model:

Task planning: agents can be applied in the decomposition of complex tasks and in the distribution to other computational resources of the split executions; agents can schedule distinct plans and strategies in order to get results in a more efficient way.

Knowledge management: acquisition, maintenance, and evaluation of knowledge is a promising field of research for both academic and business applications. Agents can access distributed, heterogeneous information sources and search for relevant information on behalf of users. Agents are a means to automate knowledge management for retrieving, analyzing, manipulating, integrating distributed information sources.
Knowledge management requires high degree of automation in order to deal with huge amount of information. Common ontologies play a central role in the description of the semantic infrastructure of information and knowledge. Ontologies require publishing and sharing services in order to provide large community of agents with common

vocabulary.
For instance the DARPA Agent Markup Language latest release (DAML+OIL) provides a rich set of constructs with which to create ontologies and to markup information so that information can be handled by software agents. These topics are being studied for the Semantic Web too.

Ambient intelligence and **ubiquitous computing** [59]: ubiquitous computing refers to information processing that is integrated into everyday objects that embed computational devices. Ambient intelligence refers to electronic environments: consumer electronics, telecommunications and computing technologies join together to develop devices that work in concert to support people that inhabit those environments in carrying out their activities (human-centric computer interaction). Intelligence is hidden in the networked devices. Agents can connect the large number of components of these real world applications to work together. Ambient intelligence is a very promising application where humans and agents could enter in a strictly coupled interaction [56]. Integration of everyday environment with computer and network technologies, requires easy-to-use interfaces between human being and accessible services that can be provided by agent technologies.

Automated auctions: internet auctions, business-to-business (B2B) applications and automated trading for stock markets are promising application fields for agent-mediated e-commerce.

Some example of research topics related to the agent paradigm are:

Automated learning: a complex task that an agent is supposed to perform is environmental adaptation, that implies learning capabilities. Learning is a key quality of agents, and research efforts are being now spent. Learning agents can be deployed in information retrieval, robotics, telecommunication and commercial areas. Trust in learning agents is still a major hurdle because they still have not achieved a consistent degree of reliability.

Interaction languages: open agent systems will require the ability to collectively evolve languages and protocols specific to the application domain. Agents with no prior experience should be able to evolve a collectively intended communication language.

Automated reasoning: one of the most challenging domain of research in multi-agent systems is the development of domain-specific reasoning models. Developing reasoning abilities for agents requires automatic inferencing procedures, hypotheses and evidence analysis and conclusion generation.

1.5 Agent-oriented software engineering

MASs have emerged as a powerful technology to face the complexity of modern application areas. They can be employed in manufacturing processes, Web services, Web-based computational markets, distributed network management. Future plans want agents to be implemented in pervasive computing, Semantic web, Grid computing. The autonomy of software agents, their high-level interaction protocols and languages, meet the modularity and autonomy requirements of decentralized applications in modern distributed systems and in dynamic scenarios.

Agent-oriented software engineering (AOSE) is aimed at proposing a variety of new metaphors, formal modelling approaches, development methodologies, modelling techniques to support the development of complex systems in an agent-oriented perspective [87].

Agents are an abstraction tool that provides a way of structuring an application around autonomous and communicative elements. When complexity arises from the interactions between the components of the system, the agent paradigm provides a natural way to model such interactions.

AOSE is mainly based on:

- modelling the environment: identification of its basic features, of the resources that it provides, and the way to interact with it
- modelling the agent society: identification of the overall rules that will drive the evolution of the system and the roles played by the agents

Several agent-oriented methodologies have been proposed over the years for the development of multi-agent systems. Actually we still are far from an unification of methods and notations. In object-oriented approach there is a universally accepted concept of object and related meta-model of the object-oriented system. In the agent-oriented approach there is not a commonly accepted definition of the concept of agent and related meta-model of the multi-agent system that represents agents, roles, behaviors, ontologies,etc. with their composing relationships.

In the following of this section we describe the most relevant meta-models of existing design methodologies. Each has its own purposes, concepts and system structures.

The *ADELFE* methodology is devoted to software engineering of adaptive multi-agent systems [8], where environment is unpredictable or systems are open. Agents are characterized by a cooperation-driven attitude, meaning that they always try to keep cooperative relations. An agent has beliefs concerning the physical environment and the other agents; this representation that may evolve, determines its behavior. Agents own some skills that are specific knowledge that enable them to realize their own partial function.

The *Gaia* methodology [81] was designed to handle small-scale closed agent-based systems. It models agents, roles, interactions, services, protocols, but misses the social aspects of a multi-agent system. Hence an an agent system is considered as a collection of roles and agents. The extension of Gaia introduced also social goals, social tasks, organizational rules. This methodology is focused on the organizational structure of the system, and on the relationships among different entities in the context of a specific organization. The central role of organizational concepts are represented by the abstractions of organizational rules and structures.

The *PASSI* methodology [21] refers to three different domains:

- the problem domain: it deals with the user problem in term of scenarios, requirements, ontology and resources. Scenarios describe a sequence of interactions among actors and the system. Requirements are represented with conventional use case diagrams. Ontological description of the domain is composed of concepts (categories of the domain), actions (performed in the domain and effecting the status of concepts) and predicates (asserting something about a portion of the domain elements). This represents the domain in a way that is substantially richer than the classic structural representations. Resources can be accessed/shared/manipulated by agents.

- the agency domain: it contains the elements of the agent-based solution. None of these elements is directly implemented. Each agent in PASSI is responsible for realizing some functionalities (roles) descending from one or more requirements.

- the implementation domain: it describes the structure of the code solution in the chosen FIPA-compliant implementation platforms. FIPA is described later in this chapter.

The *Agents & Artifacts meta-model* [65] represents the agent society analogous to human society. Agents are the autonomous proactive entities that encapsulate control and are in charge of the goals/tasks achievement. They determine the whole MAS behavior. Artifacts are the passive reactive entities in charge of providing services that make individual agents work together. Agents represent the autonomous component of the system, artifacts represents the passive components. The workspace concept represents the container for agents and artifacts useful for defining the topology of the system. Agents use artifacts to perform actions. The workspace is a dynamic set of artifacts.

Agent-oriented software engineering is supported by sets of tools for development of MASs. Among these it is worthy to mention specific MAS methodologies that lead system analysis, design and development [86, 26, 55], and the Agent UML (AUML) as specific notation technique that extends standard UML used for object-oriented design.

1.6 FIPA

The major agent standardization bodies are The Foundation for Intelligent Physical Agents (FIPA) [29] and Object Management Group (OMG) [39]. The core mission of FIPA software agent standards consortium is to facilitate the interaction and interoperation between agents across multiple, heterogeneous systems. FIPA developed specifications that defines:

- the general architecture that an agent platform should implement
- models for representing, modeling and discovering services that are available in the platform
- semantic of the communication and content languages
- interaction protocols that support transactions among agents

FIPA was established in 1996 as an international non-profit association to develop a collection of standards relating to software agent technology. It is composed of academic and industrial organizations. Their aim was to form the bedrock of a new industry by using agent technology across a wide number of applications.

At the core of FIPA are the following set of principles:

- agent technology provide a new paradigm to solve problems,
- agent technologies require standardization,
- the standardization of the infrastructure and language required for interoperation; the internal mechanisms of agents are not the primary concern

The most relevant standard produced by FIPA, dated 2002, is the FIPA Abstract Architecture that abstracts the implementation principles expressed in the FIPA2000 specification set. It specifies and defines all core architectural elements and their relationships, guidelines for the definition of agent systems in term of particular software and communication technologies and the specifications that govern the interoperability and conformance of agents and agent systems.

Abstracting the key aspect of the most critical mechanisms, it permit the creation of systems that seamlessly integrate within their specific computing environment while interoperating with agent systems of different environments. The most important components of the Abstract Architecture are contained in the gray rectangle of Figure 1.1: they provide a common point of reference for FIPA-compliant implementations, that can be developed using different technologies. The components are:

- *agent messages* written in the agent communication language (ACL), are the fundamental form of communication between agents

- a *message transport service* is the means to exchange messages among agents
- an *agent directory service* is a shared information repository where agents publish their agent directory entries
- a *service directory service* is a shared repository where agents can discover service information

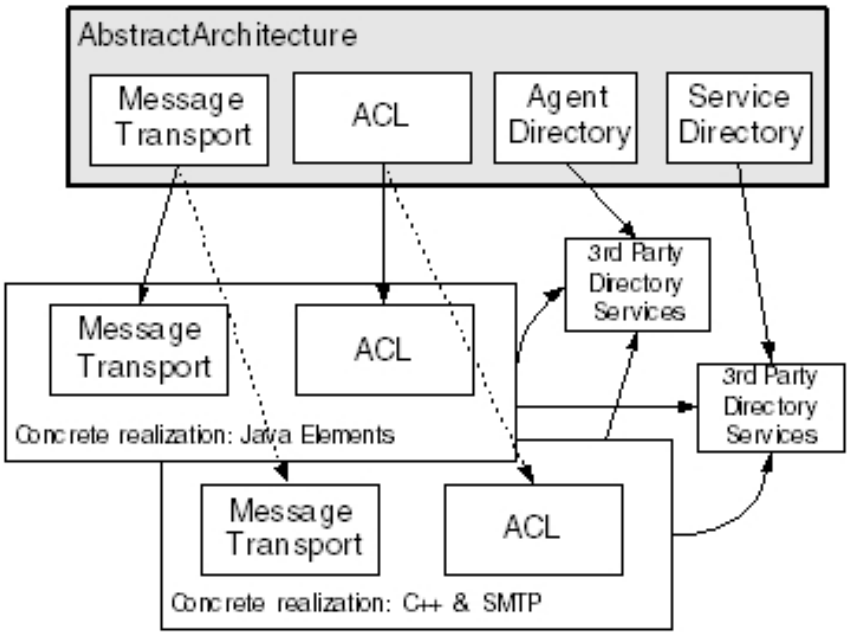

Figure 1.1: Example of concrete realization of the Abstract Architecture

Key achievements of FIPA are:

- a set of standard specifications supporting inter-agent communication and key middleware services,
- a well-specified agent communication language (FIPA ACL), accompanied by a selection of content languages (e.g. FIPA SL) and a set of key interaction protocols ranging from single message exchange to complex transactions,
- several projects outside FIPA (e.g. Agentcities) for the creation of a global network of FIPA-compliant platforms and agent application services,
- an agent-specific extension of UML, known as AUML.

1.6.1 Agent communication

Agents are fundamentally a form of distributed code processes compliant to the distributed computing model. Hence they comply with the notions of components and

connectors. Components are actors in the communication messages exchanged; connectors are means to exchange messages.

FIPA Agent Communication Language (ACL) is used to form up messages. ACL is grounded in *speech act* theory. Speech act is a technical term in linguistics and the philosophy of language: it states that by saying something, we express the intention to do something. According to speech act theory, messages in an agent system represent actions. Speech acts or communicative acts are also known as *performatives*.
An *ontology* (the term ontology has its origin in philosophy) is defined as a formal representation of a set of concepts within a domain and the relationships between those concepts.
The combination of speech act and ontologies results in a precise intended meaning of complex interactions.

FIPA defines a communication stack that is composed of the following sub-layers:

- **transport**: message transport protocols defined by FIPA are IIOP, WAP, HTTP
- **encoding**: messages are encoded in XML, Bit-Efficient, or String
- **messaging**: FIPA defined parameters that a message has to contain in addition to the payload (the content of the message)
- **ontology**: FIPA messages can be compliant to a specific ontology that has been fixed a priori by the interacting parties
- **content expression**: this layer highlights the content languages used
- **communicative act**: this layer classify the message according to the action or performative it implies
- **interaction protocol**: it defines the specific interaction protocol used in the communication

Common acts used within the communication capture the essence of most forms of basic communication and are for instance:

- *inform*: the sender informs the receiver that a given proposition is true
- *request*: the sender requests the receiver to perform some action
- *agree*: the action of agreeing to perform some action
- *not understood*: the sender informs the receiver that it did not understand the action that was performed
- *refuse*: the action of refusing to perform a given action

Based on these communicative acts, FIPA defined a set of interaction protocols that consist of a sequence of communicative acts for multi-message action coordination. For instance the Contract Net protocol starts with an agent's call for proposal with specific conditions issued to one or more agents that are potential contractors. Each of them can refuse or propose an action. The initiator of the contract can accept or reject the propose, and if accepted the participant informs the initiator when it performs the action.

FIPA ACL does nor mandate any content language for expressing the message. FIPA SL, XML are example of content languages.

1.6.2 Agent management

The second fundamental aspect of agent systems addressed by FIPA specifications regards agent management. It establishes the logical reference model for the creation, registration, location, communication, migration and operation of agents.

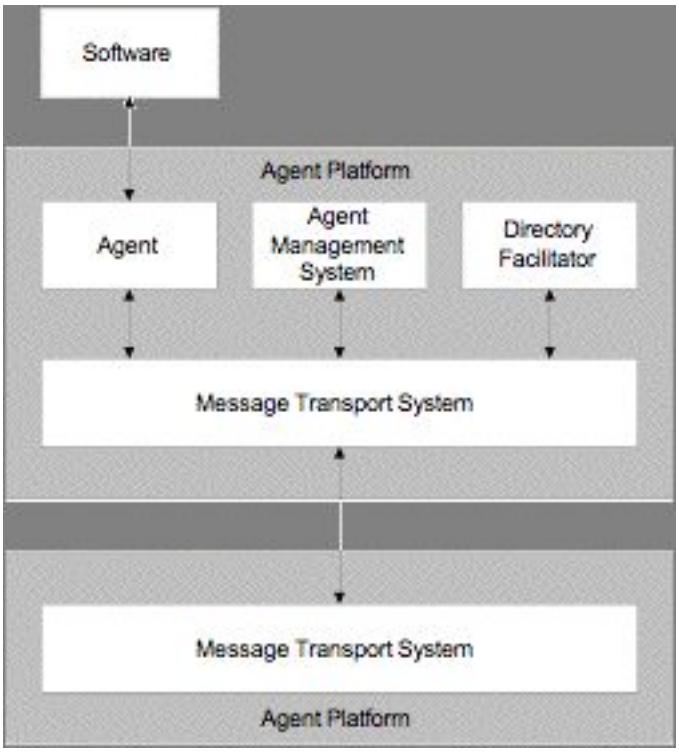

Figure 1.2: Agent management reference model

The specification defines a normative framework within which FIPA-compliant agents can exist and operate. The agent management reference model consists of the components depicted in Figure 1.2. In details, the framework is composed of:

- **Agent Platform**: the AP provides the physical infrastructure that host agents and their deployment. It consists of the machines, the operating systems, the

FIPA agent management components, the agents themselves. A platform may be spread across multiple computers. The specific internal design of a platform is left to the developers.

- **Agent**: an agent is the computational process that run on a platform; it can offer services (*capabilities*) that are published in the directory facilitator; it must have at least one owner and support the notion of identity that can be described using the FIPA agent identifier (AID); an agent may be registered with more than one transport address. FIPA defines just the structure and the encoding of agent messages.

- **Directory Facilitator**: the DF is an optional component of the platform that provides yellow pages services to agents. To publish its service an agent is supposed to register with a DF. A platform can support any number of DF, organized in a federation of DF.

- **Agent Management System**: the AMS is a mandatory component of the platform; it manages the operations of a platform, for instance the creation and deletion of agents, the migration of agents from host to host within the same platform or migration from a platform to another. Each agent must register with the AMS it is running on in order to obtain its AID. Only one AMS can exists in each platform; if the platform spans multiple machines the AMS is resident just on one machine but serves all of them.

- **Message Transport Service**: the MTS is a service provided by the platform; it manages all the messages sent or received by agents in the platform.

In Figure 1.2 the light gray rectangle represents an agent platform with the components described above. The platform communicates with another platform (the bottom one) and can interact with other external software that is not necessarily part of the agent system (e.g. software that allows a device to an agent).

1.7 JADE

JADE [1, 6] is a software platform that provides basic middleware-layer functionalities which are independent of the specific application. It provides software tools for the realization of distributed applications that embed the agent abstraction. It is implemented in Java language and has full compliance with the FIPA specifications. It was distributed in 2000. Its key features are:

- efficient transport mechanism for asynchronous messages,

- implementation of both Directory Facilitator (DF) service (yellow pages) and Agent Management System (AMS) service (white pages),
- support for agent mobility: an agent can migrate between machines in the sense that its code is transfered to another AMS for execution,
- an effective agent life-cycle management,
- a set of graphical tools to support debugging and monitoring,
- support for ontology development,
- a complete library of interaction protocols,
- integration with web-based technologies.

A JADE platform is composed of several agent containers (Figure 1.3) that can be distributed over the network. Containers are the java processes that provide the JADE run-time and all the services needed for hosting and executing agents. There is a special container, called the main-container, that is the first container to be launched in the platform and all other containers are joined to the main by registering with it. The main container manages the registry of all agents present in the platform, it hosts the agent management system AMS, the directory facilitator DF. A platform is composed of at least the main container; other containers can join it.

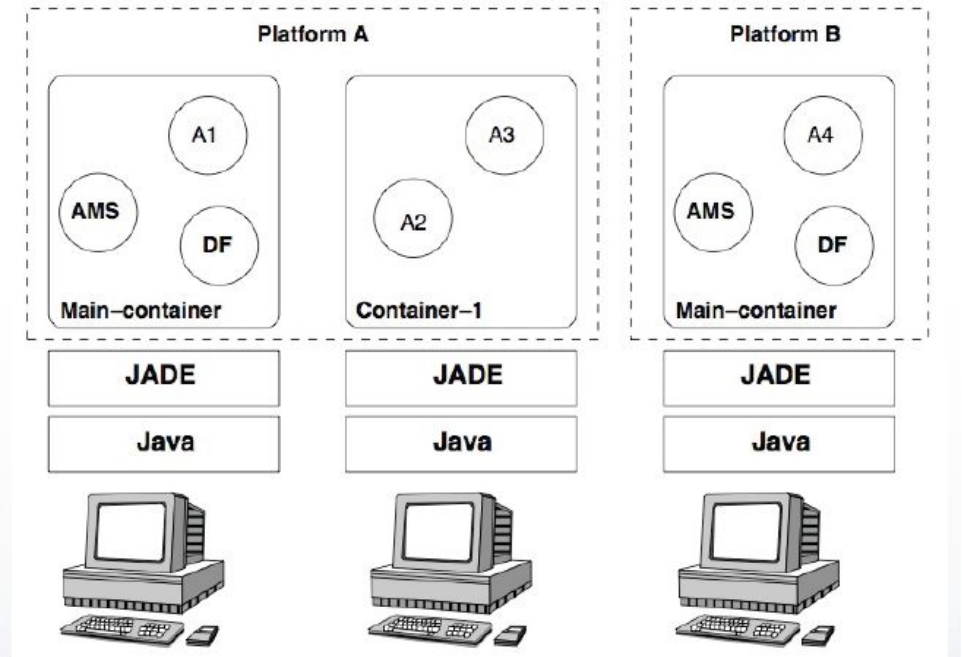

Figure 1.3: JADE architecture: platforms, containers and services

JADE implements all the standard Message Transport Protocols defined by FIPA. For message to be dispatched among agents living internally to the same platform,

JADE uses a proprietary transport protocol called Internal Message Transport Protocol (IMTP), that is based on Java RMI, and is not compatible with FIPA standards. For messages exchanged between agents living in the same container, JADE uses Java events.

Agents' tasks are represented with "behaviors". An agent can have multiple behaviors. Scheduling of behaviors in an agent is cooperative and not pre-emptive, i.e. when a behavior is scheduled for execution, it runs until it returns.
When a behavior returns, it is pushed in the queue of agent's behaviors, and the next in the queue is picked for execution.
The programmer is responsible for the correct switching from one behavior execution to another, not the AMS.
Figure 1.4 shows the scheduling of the behaviors: in the `setup()` method all the behaviors are added to the agent; a behavior `b` is started calling its `action()` method, and it is removed from the queue calling its `done()` method.

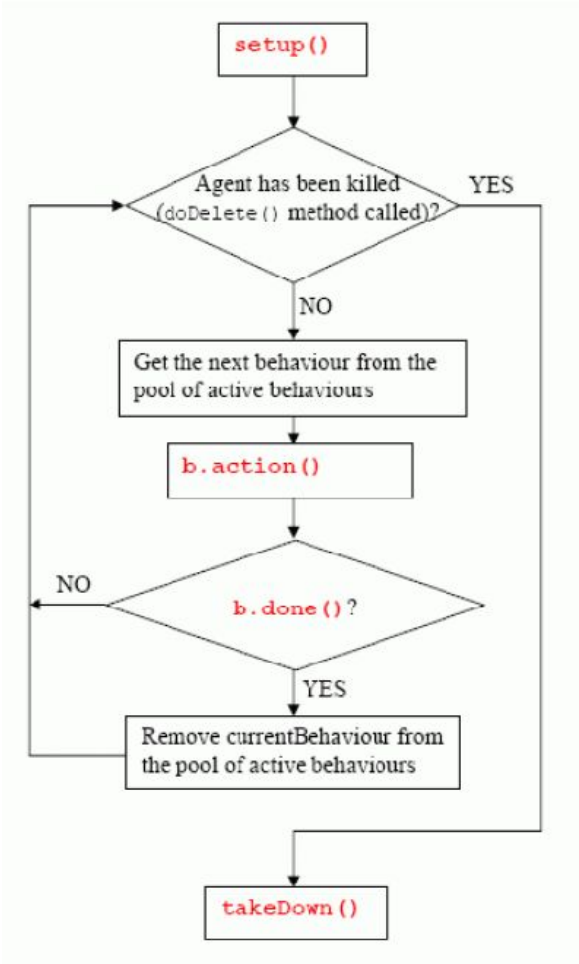

Figure 1.4: JADE thread execution

The advantages are:

- an agent requires just one Java thread because only one behavior at a time is executed (light computational load),
- behavior switching is faster than thread switching
- no synchronization is required for concurrent behaviors of the same agent that access the same resources because all behaviors are executed by the same thread: it results in improved performance,
- at behavior switching the status of an agent does not include any stack information, and this allows to save the status of an agent hence the implementation of advanced features, such as agent mobility (the agent is transfered to another container) and agent persistence (the agent is stored for later resumption).

JADE can also be installed in mobile resource-constrained devices such as Personal Digital Assistants (PDA) and and mobile phones, with reduced processing power. JADE-LEAP (Lightweight Extensible Agent Platform) is the project that developed a light version of JADE that can be used in small devices, supporting the interaction with JADE platforms embedded in larger devices.

1.8 Security issues in agent systems

Any IT system is affected by security issues that regard communication. MAS are strongly based on communication among agents. Communication security is protected with mechanisms that provide:

- **confidentiality**: information can not be accessed by unauthorized entities
- **data integrity**: information can not be manipulated by unauthorized parties without being detected
- **authentication**: information origin can be assessed

Security is traditionally protected with cryptographic mechanisms, such as encryption, decryption, public key, private key, secret key, digital signature. However these mechanisms are expensive from the computational perspective and they do not fit with highly dynamic agent systems (entities enter and leave the system in an unpredictable fashion).

Mobile agents can also migrate during their execution among hosts, in order to move computation in a more suitable environment, due to the presence of required resources or services. Mobility introduces significant security issues that require to be properly addressed before agent paradigm may be applied in future technology.

Agents running on a platform are potential threats because they are owned by different users with different aims. Before allowing an agent to run on an host, the platform should check if the agent's intentions are either fair or malicious, but platforms do not have such a mechanism able to verify the content of an agent and analyzing its code. Intrusion detection systems use specific technology to perform code analysis, but it is not applicable to agent systems because of the high number of agents that a system could manage.

Actual implemented agent systems perform access control through agent identity checks. Other information that regards the incoming agent can be checked too: origin host, previously visited hosts, owner identity, etc. Agent systems employ authorization policies that associate agent identities with permissions: once an agent has been authenticated, it is given authorization to perform a limited number of operations in the platform according to the policy.

Access control mechanisms are not trivial because there is not a central authority in agent systems that can be queried for identity or other information verification.

Security issues are more relevant in open systems. Open system are composed of resources from several systems distributed over the network in order to serve a wide range of users. Open systems do not have a complete knowledge of all their components and participants. Bound closed system security can be easily solved by authorizing only internal known users. Agents and open system are analyzed in Chapter 3.

Actual implemented agent platforms target communication security and authentication (e.g.: Aglets [66], JADE [45], D'Agents [37], SeMoA [10]). This kind of security does not fit open systems because secure communication between parties without pre-established relationships becomes cumbersome.

From a more general perspective, security issues in mobile agent systems affect both agents and hosts. Malicious hosts threat agents that are running on them because the host has the complete control of agent execution. The host can modify the code and data contained in the agent for its advantage. For instance it could steal electronic money from the agent; it could modify the code in order to force the agent to behave in a specific way and to operate incorrectly. Actual solutions for protection from malicious hosts are aimed at detecting rather than preventing [13]:

- contractual agreements among agents and hosts for a fair treating; but it is not a trivial task to prove broken agreements;

- hardware provided by a trusted third party can ensure a safe execution environment for agents; the trusted hardware could be used just for sensitive tasks, such as handling of the private key in decryption operations, and not for the entire execution of the agent;

- setting up trusted nodes, where agent can migrate to safely perform sensitive operations; for instance when the agent decides to purchase one of the offered items it can migrate to a trusted node and request to perform the purchase, avoiding to pass sensitive information to the seller that could be malicious;

- tracing execution for detection of unauthorized manipulation of the agent's tasks: each platform is required to create a log of the agent's performed operation; a drawback is that the agent could be required to manage big size logs;

- partial result authentication codes: the agent status is encrypted during its execution in a remote host; each stored status is encrypted and can be later checked in order to know its modification history and to detect malicious modifications applied by a visited malicious host;

- partial results can be encrypted with an asymmetric algorithm and sent back to the agent owner that collects partial information along the agent's route; the agent is required to carry just the public key for encryption;

- the agent can carry encrypted data that could be decrypted only when specific conditions are met, hence only certain hosts can know agent's sensitive information;

- it is possible to encrypt some functions that the agent will perform; the host is able to execute the encrypted function, but can not understand the meaning of it; the outcome of the encrypted function is the encryption of the original outcome; agent execution code can be kept secret from the executing host; research is focusing on encrypted functions because not all function can actually be encrypted.

Protection of the agent platform from malicious agents can be achieved with the following mechanisms:

- sandboxing: it separates processes into distinct domains: untrusted processes are executed in virtual address space and can not interfere with other applications; access to system resources can also be controlled in such a way. Java utilizes sandboxing, so it is a suitable programming language for MAS implementation;

- proof carrying code: the author of the agent is required to prove that the agent code is not malicious; it is difficult to generate such a formal proof, so this solution has limited applications;

- code can be signed, so the platform can check if the code has been manipulated from previously visited platforms;

- the agent can provide its path history to the hosting platform in order to let it know if the agent visited an untrusted platform that could have manipulated its content;

- the agent can ensure the hosting platform that its state and code were not tampered by previous visited hosts. This is done including an appraisal function created by the agent owner that becomes part of the agent code. The hosting platform uses the appraisal function to check if the agent state and code have been manipulated. It is not of easy implementation because it requires to capture the state of the agent.

1.9 The challenge of MAS

Research on intelligent agents is related with a wide range of sub-disciplines of information technology, including computer networks, software engineering, artificial intelligence, human-computer interaction, semantic web, distributed and concurrent systems, mobile computing, control systems, electronic commerce.

Among these disciplines agents can play a key role in those that require flexible, autonomous mobile components that compose the system, for instance in mobile computing (computing performed with mobile computers such as wearable computers, personal digital assistants and enterprise digital assistants, smartphones, carputer (computer platforms installed in vehicles), ultra-mobile personal computers).

The basic idea of software agents and their potential applications are quite appealing. Agent technology has not still move into mainstream computing technology because of the lack of theories that are necessary to base the development on: common methodologies for agent-oriented analysis and design, agent design patterns, validation and testing techniques, tools for agent systems development process [57].

Actual agent applications are developed in an ad hoc fashion without any rigorous design methodology that supports the definition of the agent system specification details, and that addresses functional properties (behavior of the system) and non-functional properties (scalability, robustness, reliability of the system) of the developed system.

Developers require libraries for the implementation of organizational models (e.g. hierarchical or flat), agent models (reactive, deliberative), task models (monitoring, information filtering, carrying out transactions), communication patterns, ontology patterns, interaction protocol patterns.

Agent Unified Modeling Language (AUML), an extension of UML, is an example of tool that supports developers of agent-based application.

Furthermore, different agent system developers emphasize different aspects, so actual agent platforms and languages differ in several aspects turning out in integration

problems.

Mobile agents represent a new approach in the interaction among users and online services. Mobile agents can move and roam in heterogeneous networked hosts. On the one side agent technology is very useful and powerful, from the other one they represent a security issue: system administrators will be unlikely to adopt the mobile agent paradigm unless security concerns are fully addressed (protect host from malicious agents, and agents from each other). Actually agent systems do not provide tools that can ensure security for agents and hosts.

Collaboration among agents in open environment requires high trustworthiness among agent owners. Actual platforms do not implement any mechanism for the definition and the creation of trust relationships among agents and this represents a recent topic in which research is focusing its attentions.

Agents are a solution for integrating agents into the existing Web infrastructure [58]. Agent standards does not completely link to existing industry software standards, but there are efforts to integrate agents and W3C Web services communication standards.

Agent technology will probably become part of the ordinary computing. Comparing agent oriented technologies (AO) with object oriented technologies (OO), we realize that after two decades of OO (1982), Java was still 13 years away from public release. Probably agents will not have the same impact as OO, but it is not too late to see an effective employment of AO technologies.

Once dynamic and open systems spread and become ordinary computing solutions, agents will integrate with IT mainstream. Agent paradigm will be adopted as fundamental and ordinary, not just a conceptualized domain of research [61]. Agent technology is not spreading in a big-bang fashion, but slowly agents are turning up in IT systems.

1.10 Contribution and organization of this thesis

This thesis describes the design of a reputation management model for agent systems. Reputation allows a society of agents that are part of a community of platforms to collect trust opinions about competence and honesty of interacting agents. Opinions are then combined in order to form agents' reputation evaluations.
These evaluations can be used by agents that are searching for information about unknown potential interaction partners.

We adopted an unusual centralized approach for agent systems that is aimed at creating context groups where agents are bound by a common context that is related to the trust evaluations provided by participating agents.

The model is also composed of a layer of informative points that is aimed at sup-

porting an efficient information distribution. This layer constitutes a social infrastructure that binds the context groups and that supports reputation information collection.

The reputation system is decoupled from the agents, in fact agents are not required to implement specific functionalities to join the system: they just need to share the same communication ontology.

The thesis is structured as follows. Chapter 2 introduces the characteristics of trust and reputation from both social science and computational perspective, surveying actual implemented major trust and reputation management systems. Chapter 3 describes the reputation system model design for open multi-agent systems and its features. Chapter 4 gives details of the social infrastructure based on the components of the reputation systems and defined over the society of agents.

Chapter 2

Computational trust: concepts and state of the art

2.1 Introduction

E-commerce, virtual marketplace, consumer-to-consumer applications (C2C), web auctions are increasingly used by people that offer goods and services for sale. They offer temporal and geographic advantages respect to traditional marketplaces. Electronic transactions are still based on the human notions of trust and risk in decision-making. In a virtual marketplace people usually do not physically meet to complete transactions, they remain anonymous, so new risks potentially raise up.

Transaction actors have to trust each others. In order to determine trustworthy actors, virtual marketplaces embed mechanisms that provide reputation information. This information is based on feedbacks (e.g. opinions) provided by people who participated in transactions that regard other participants.
Transaction actors evaluate the expectation or the belief that the partner will honestly and cooperatively behave before getting involved in interactions, attempting to limit threat exposure.

Reputation systems were firstly introduced and employed in electronic commerce transactions, e.g. eBay, Amazon. Computational concept of trust is usually applied between partners in business transactions [23]. Trust is the belief regarding honesty, reliability, willingness of another entity. Reputation is an overall summary of a collection of thousands of individual opinions. The mean average of these opinions are aimed at detecting relevant patterns of users' behavior.

2.2 Open systems and security issues

Distributed computing system are aimed at connecting users and resources in a transparent, open, and scalable way. *Openness* implies that each subsystem is continually available to interact with other systems. In general, IT moved from the development of closed and self-contained systems to the definition of new models for the development open systems. To figure out what an open system should be, we can represent it with the society of human beings, that is strongly based on communication, interaction and cooperation.

Open systems are characterized by dynamically changing sets of users that autonomously join and leave the system in an unpredictable fashion. These systems are referred as open because they do not have clearly defined boundaries that limit the interaction among entities. Hence open systems require a new approach for what is concerned with interaction among entities because the resources and the participants that compose the system continuously enter and leave the system.

Modern open distributed systems are required to support dynamic interactions with previously unknown entities and to provide some kind of social rules that govern high-level interactions. Open systems should be able to reorganize themselves to deal with critical situations such as unstable connection (e.g. for mobile computing), node failures (e.g. P2P systems), security threats both for entities providing services to unknown requesters and for users who access services of unknown providers. Security threats require new solutions because resource control policies in local contexts do not fit any more with open systems.

Distributed and pervasive computation applications, networked computation deployment and web applications ask for new design methods and new programming paradigms that support effective interactions among participating entities in order to exploit the potentials of open systems.

The *agent-oriented* (AO) programming paradigm looks a promising tool for the design of emerging requirements in IT even if it is not widely employed in the design of computing systems. Multi-agent systems are characterized by autonomy and migration capability that fit very well with open system features.

2.3 Trust and reputation features

The formal and general definition of trust in any dictionary says that it is a confidence in or a reliance on some quality of a person or thing.
Several areas contribute at the definition of trust concept, e.g. sociology, psychology and philosophy. The model for computational trust is derived from human trust formation process. Some key features of human trust concept are risk assessing, trustworthiness determination, decision-making process, reputation formation, evidence detection

[36].

Gambetta's definition [33] contains a complete description of the aspects of trust: *"when we say we trust someone or that someone is trustworthy, we implicitly mean that the probability that he will perform an action that is beneficial or at least not detrimental to us is high enough for us to consider engaging in some form of cooperation with him. Correspondingly, when we say that someone is untrustworthy, we imply that the probability is low enough for us to refrain from doing so."*.

The concept of trust is involved when in an ambiguous situation, an entity is required to evaluate if the outcome of the interaction with another entity is perceived to be beneficial or to be harmful. The uncertain evolution of the interaction implies that the entity is supposed to make decisions based on trust information. Trust is not taken into account when the entity clearly knows how the interaction is evolving. Uncertainty is caused by the unavailability of complete information regarding the outcomes of the interaction.

Trust is *subjective* because the way an individual forms belief is based on his own disposition. Trusting decisions require also a cost-benefit analysis: the entity weighs the cost and benefit involved in the situation. Risk evaluation is fundamental for decision-making, hence information related to the involved risks is necessary in trusting decision process. This requires observation and evidence gathering.

From the social perspective, trust is strictly related to the context in which the decision is being made, hence trust is *context-specific*. Context assumptions are necessary to describe the environment where the situation is evolving, and the roles of the entities that are involved in it. Environmental features affect trust and risks assessments. Roles and context information are useful for a better analysis: for instance if a taxi driver is trusted as an excellent car driver, we probably do not trust him to fly an aircraft.

From the computational perspective, we refer to Grandison definition of trust [34]: *"trust is the belief in the competence on an entity to act dependably, securely, and reliably within a specified context"*.

Taking into account all the above listed characteristics of trust, it can be described as a mean to reduce complexity in decision-making regarding uncertainty in interaction with unknown entities. In trusting decisions entities fix a threshold level of trust to distinguish trust values that lead to a positive decision from those that lead to a negative decision. In this process, entities employ optimist or pessimist strategies.

Trust has an *asymmetric* nature, i.e. for each trust relationship there is an origin and a target (the trustor and the trustee) who are distinct.

Trust is based either on *direct experience* with the entity, or on *indirect experience*, that is based on the knowledge related to interactions that third-parties experienced with the entity.

Recommendations are used to give a transitive property to trust: recommendations are indirect evidence of trustworthiness. Trust is formed upon combination of

direct observation and third parties' recommendations. Recommendations suffer of the integrity problem, meaning that also recommendations should be subject to trusting process because they are affected by honesty, reliability, competence of its sender.

2.4 Trust measures

People has an intuitive concept of trust, meaning that usually human beings do not assign to trust a numerical value. In a computational environment, it is necessary to evaluate trust in a numerical value.

A trust *origin* evaluates his trust degree on a trust *target* for a given trust purpose analyzing available evidence and information. Several factors can contribute to this evaluation: the context, environmental factors, time, roles of the two entities, integrity of eventually received recommendation, subjectivity, target's disposition to risk.

We now describe some models that are used to quantify trust from the computational perspective.
Trust as a *probability* measures the probability that the target will behave in a reliable way for a specific purpose. This approach does not allow an easy combination of measures with ordinary mathematical tool, e.g. multiplication for transitive trust. This model can not capture uncertainty too.

Trust measurements with *Bayesian* methods use the beta probability density function [54]. The method considers the numbers of positive observed behaviors α and the negative ones β. The trust measure is calculated based on a priori measure (α, β). The beta probability density function represents the probability that an event has a certain probability value to occur given α and β. It captures uncertainty, even if the trust measure is not easily comprehensible.

Trust can be measured with the *belief functions* of Dempster-Shafer theory [73].
A belief function assigns probability values to sets of possible outcomes. This approach is related to probability theory, but outcomes do not necessarily add up to 1. Belief calculus and subjective logic [50] are used when information is incomplete.

An "opinion" is a metric to express belief, and it captures the entity's belief in the truth of a proposition regarding a trust target for a trust purpose. An opinion is expressed as belief, disbelief and uncertainty: (b, d, u) where $b + d + u = 1$.
For instance if agent A interacted with agent B, and B behaved positively 8 times, negatively 1 times, and 1 time the outcome is not certain, the opinion of A about B regarding trust purpose x is expressed $\omega_x^{AB} = (0.8, 0.1, 0.1)$.

Recommendations are opinions passed to other agents. These are appropriately discounted, according to distance semantic and trustworthiness of the recommenders. Discounts and combination of opinions are made using subjective logic: for instance if ω_x^{AB} is A's trust opinion about B, and ω_x^{BC} is B's trust opinion about C, we have

that $\omega_x^{ABC} = \omega_x^{AB} \otimes \omega_x^{BC}$ is A's trust opinion about C obtained by discounting B's recommendation about C and considering the opinion that A has about B.

For simplicity some models express trust using *discrete values*, e.g. "completely trusted", "trusted", "not at all trusted" and "very untrusted".
The main issue with this approach is to perform mathematical operations on these values, weighting, discounting, combining values. A solution is to map from the discrete domain to numerical domain, perform the computation and then convert it back to discrete domain:

$conv : \{comp.trusted, trusted, nottrusted, v.untrusted\} \rightarrow \mathbb{Z}$
$conv(discreteOpinion) = numericalOpinion$

$f(numericalOpinion) = numericalResult$ (f performs the required computation)

$conv^{-1}(numericalResult) = discreteResult$

2.5 Trust models

The trust management approach for security issues in large open distributed systems was introduced because of the inadequacy of traditional authorization mechanisms (authentication and authorization for access control).
Rigid cryptography mechanisms for precise authentication of entities are frequently used in systems that employ access policy specifications in order to protect integrity and security from malicious entities.
Trust approach takes into account relationships that bound agents in their interactions. In [68] the two different approaches are defined *hard security* for traditional mechanisms like authentication and access control, *soft security* for social control mechanisms based on trust concept.

A trust management system is required to support users in assessing trustworthiness of remote entities afterward evidences about unknown transaction partners have been collected. Trust is evaluated and it evolves based on collected evidence of past behavior that has been directly observed or recommended by other entities. Trust threshold is subjectively defined by each trust origin and it depends on the level of risk it takes.

Trust-based decision-making depends on the predictability of entity's behavior; if it dynamically changes, trust is not a reliable measure to be used in decision-making. Now we describe some trust model and management systems.

PolicyMaker [12] takes as input a set of local policy statements, a collection of credentials and a set of proposed trusted actions; the query engine evaluates whether

the credentials prove that the a requested action complies with local policy. The engine uses a general purpose, application-independent algorithm for checking proof of compliance.

KeyNote[11], the successor of PolicyMaker, implements more features, such as signature verification, the use of a specific policy assertion language. Its responses to the queries are application-defined string, that provide advices to the requesting application.

The Rule-controlled Environment For Evaluation of Rules and Everything Else (REFEREE) query-engine [22] is based on PolicyMaker. It interprets trust policies and returns a statement list that provides the context for the decision and justifications for the answer.

Simple Universal Logic-oriented Trust Analysis Notation (SULTAN) [35] is a computational framework for trust management that collects, analyzes, provides evidences related to competence, honesty, dependability in order to make assessments regarding trusting relationships. It is composed of a specification editor used to describe trust relationships using the SULTAN policy language; an analysis tool to check the consistency of the trust and recommendation statements; a risk service that retrieves risk information and provides risk probabilities; the monitoring service that updates information when agents get involved in new interactions, hence the system adopts a dynamic concept of trust.

Josang uses belief theory and subjective logic in his trust model [49]. Belief expresses an expectation of how an entity will behave based on direct experiences, recommendations from other sources.
The opinion of an entity A that captures information about an entity B regarding a specific trust purpose x, is $\omega_x^{AB}(b, d, u)$. b represents the belief in B trustworthiness, d is the disbelief, u is the uncertainty, where $b + d + u = 1$ and $\{b, d, u\} \in [0, 1]$.
A trust measure is the expected probability of ω_x^{AB}. The model provides a formal process for decision-making, that includes the notion of risk by assessing utilities to the possible outcomes of a transaction.

Trust model of Marsh [60] is based on trust concepts from psychology, philosophy, sociology. The model is based on the definitions of:

- knowledge $K_x(y)$ that agent x has about agent y,
- the basic trust $T_x \in [-1, 1)$ that represents the agent's overall beliefs of the world,
- the general trust $T_x(y) \in [-1, 1)$ that an agent x has in agent y irrespective to a specific situation,

- the situational trust $T_x(y, \alpha) \in [-1, 1)$ respective to the specific context α,
- the utility of a situation $U_x(\alpha) \in [-1, 1]$ and the importance $I_x(\alpha) \in [0, 1]$ of a situation.

These values are used in a complex calculation of the *cooperation threshold* that represents the minimum trust to engage interactions with a trust target.
This model does not take into consideration recommendations.

Trust model of Abdul-Rahman and Hailes [3] considers interpersonal trust among a source and a target for a specific context. It aims at assisting users in the identification of trustworthy entities within a virtual community and at giving autonomous agents the ability to reason about trust. Trust measures are based on prior direct experiences with the target agent and on third parties' recommendations. Evidences for trust evaluation are stored in four different sets: the set of direct trust experiences, the set of recommended trust experiences, the set of known contexts, and the set of agents with which previous interactions have occurred. The model does not include any trust-based decision-making process. This is left to the user's own disposition to determine interaction policies and risk evaluation.

Secure Environments for Collaboration among Ubiquitous Roaming Entities (SECURE) [15] takes into account concepts related to human trust: relation among trust and risk, subjectivity, transitivity of trust through recommendations or reputation, integrity of entities in recommendations, context and situation dependence of trust, evidences for trust assessing). Its trust model is general and can be applied to a variety of applications.

The model is based on a trust module that processes trust evaluations. It is composed of four modules: the trust module that computes trust values; the evidence manager that manages and stores observations, recommendations and other information; the risk module that evaluates risks based on trust values, context information and transaction utilities; the decision-making module that handles interaction requests. The model can be integrated with user defined policies too.

Evidence of occurred interactions are classified in one of the following rates:

- supporting,
- inconclusive,
- or contradicting

The trust model provides a way to model evidence that may be observed about a given interaction. Trust values are derived from this evidence, hence a trust value is a triple $(s, i, c) \in \mathbb{N}^3$. Trust values are a function of evidence values and range in $[0, 1]$.

The interpretation is that out of $s + i + c$ interactions, s interactions support the occurrence of a configuration x of a particular event structure (e.g. a given outcome of the interaction), i interactions are inconclusive about the occurrence of x, and c interactions contradict x.

Triples can be modified by third parties' recommendations, considering their respective trust values: a trust value is a combination of direct and indirect experiences. Integrity checking process is not implemented in the model.

Reputation concept is not considered in this model, indeed trust is evaluated with direct experiences and recommendation.

2.6 Reputation management systems

Reputation is a collection of recommendations which are aggregated to form a measure of an entity's character with regard to ability or reliability in interaction. A *reputation system* is built up to accumulate and distribute recommendations in order to allow agents to use the information to make decisions about whether to trust another or not. A reputation system requires the existence of a social network of agents that provide opinions, recommendations.

Reputation systems can be centralized or distributed.

- **centralized**: a central service collects recommendations, aggregates them and derives reputation measures, that are provided to members of the social network, using a specific communication protocol. Centralized systems are actually used for e-commerce (auctions and online stores);

- **distributed**: the communication protocol is distributed, and each entity has his own method to derive reputation from collected recommendation from peers.

As described in [53], there are different types of recommendation aggregation methods and evaluation mechanisms for reputation measures computation:

- simple average: for instance eBay [27] uses simple summation of positive and negative feedbacks (a unique feedback considered for each user): the method is very simple but obviously very inaccurate because it does not take into consideration integrity of recommendations and subjectivity of feedbacks; another example is Amazon [4] that measures reputation as an average of all recommendation scores;

- Bayesian method: these systems compute reputation scores by statistical updating of beta probability density functions: the updated reputation score is obtained by combining the previous reputation score with the new rating; the reputation score can be represented in the form of the beta density function or its expectation value [5];

- discrete methods: for instance in [19] possible trust values and their ordering are determined at the setup of the application according to the possible outcomes of the interactions; for instance trust value can be represented by a pair (p, n) of integers that represent the positive and negative interactions experienced with an entity; each interaction increments the pair; a trust policy determines if an entity is trustworthy or not according to the value of the pair;

- belief functions [73]: they are used to obtain a degree of trust that regards an entity from subjective opinions based on independent evidences; they use the subjective independence to determine joint probabilities for various possibilities of opinions reliability; they have been implemented in [49];

- fuzzy models: fuzzy concepts are used to represent trust and fuzzy logic is used for trust reasoning [69].

The decentralized reputation system of Abdul-Rahman and Hailes [3] is based on the propagation of information as recommendation around the system. A recommendation contains the identity, the related trust context and the discrete trust value. Entities provide recommendations when queried. Recommendation are weighted according to semantic distance. Combination of collected discrete trust values are done with heuristic methods.

Mui's distributed reputation system [64] considers recommendations as boolean ratings, hence an agent recommends another one with approval (1) or disapproval (0). Context is defined as a set of boolean values that describe the presence (1) or not (0) of specific attributes. Recommendations are propagated with queries from an agent to another agent. Recommendation integrity is considered by weighting the ratings in reputation formation according to a list of most reliable and preferred recommenders. Recommendations are combined through a Bayesian calculation. Trust is represented too simplistically as a combination of binary values that are the outcomes of interactions (cooperation or defection). The model focuses more on recommendation propagation than on reputation formation.

In Jøsang's reputation engine called Beta Reputation System [54] evidence of occurred interactions are represented as a couple of satisfaction and dissatisfaction ratings (r_A^B, s_A^B) by entity A about B. Ratings are stored in the central engine, summing all the r values and all the s too. Reputation is expressed as an expected probability measure about entity's likely future behavior. Reputation is modified with Bayesian techniques when new evidences are received. Belief metrics weight the reliability of recommenders and their integrity. Time is taken into account with a forgetting factor applied by an algorithm that perform the discounting. This model lacks parameters for trustworthiness formation and for context specification.

Buchegger's distributed reputation system [17] was developed for P2P networks. The model is based on opinions $R_{i,j}$ formed by an entity i about j regarding its behavior in file-sharing activities. $T_{i,j}$ is the integrity opinion by i about j. For each couple of entities a summary record $F_{i,j}$ of occurred interactions is stored. $T_{i,j}$ is updated every new opinion with a Bayesian process. A fading algorithm degrades reputation after a period of inactivity. This model does not capture trustworthiness aspects for trust formation. The contexts that are related to the interactions are not considered in this model.

Yu and Singh's reputation model for e-commerce agents [84] is a decentralized system with distribution of recommendation through query-response.
Reputation is formed upon trust assessments by i about j at time t: $T_i(j)^t$. Trust is based on direct past experiences and recommendations. Recommendation collected from a referral chain, that is the path that a recommendation follows from an agent to another before getting to the requester of the recommendation, is weighted depending on the integrity of the agents in the path.
Evidence for trust formation are combined using Dempster-Shafer belief theory [73]. Context factors are not taken into account, whereas time is considered.

Xiong and Liu proposed PeerTrust [82] distributed reputation management. Recommendation are weighted taking into account the source reliability, the transaction and community context, temporal factor. Feedback of a transaction is requested by the system from the two involved actors. Two methods are employed to calculate credibility of source recommendation: one is based on the trustworthiness of the source within the community, and the other is based on personalized similarity or semantic distance between two peers. The trust information is mono-dimensional, hence very simplistic.

ReGreT [70] reputation model focuses on social relations. It is implemented for a simulated supply chain. It distinguishes three types of relationships among agents: cooperation, competition and trade, that is compatible both with cooperation and competition.
Reputation ratings have three dimensions: individual (direct interaction evidence), social (composed of recommendations from witnesses which interacted with the target agent, from target's neighborhood reputation and system reputation) and ontological (that combines reputations on different aspects to calculate complex reputations).
Recommendations are combined using fuzzy rules. ReGreT adds a social structure to the typical reputation concepts, and introduces a multi-faceted concept of reputation.
This model cannot be easily applied to a real-world system when it is characterized by complex dynamics.

FIRE [47] trust and reputation model was developed for open MAS. It introduces four different types of information.
Interaction trust is derived from past experience of direct interactions.

Witness reputation is based on collected evaluations of other agents' previous experiences with the target agent.
Role-based trust is based on predetermined trustworthiness between evaluator and target derived from rules of the specific domain, or roles of the two agents.
Certified reputation is based on third-party references provided by the target agent.
The overall trust value is calculated as a weighted average of the four values; the weight is an evaluated reliability value of each information. The model produces a reliability value that expresses the confidence of the trust model in producing each trust value. Ratings are expressed as tuples: $r = (a, b, c, i, v)$ where a is the evaluator agent, b is the target agent, c is the term of trust, i identifies the interaction, v is the rating value.
The model assume that agents are willing to share their experiences with others, and that agents are honest in the information they provide.

Chapter 3

Context-dependent Reputation Model for Agent Systems

3.1 MAS and open systems

Implemented MAS platforms employ *hard* security mechanisms that assure confidentiality for communication and authentication for incoming agents. This is effective for closed systems, where only known agents are allowed to enter.
In open computer networks, interactions among entities create virtual ad hoc cooperative networks where pre-established trust between actors is not required.

Thanks to the high availability of communication mechanisms, computational systems can easily increase and improve their interaction with remote systems (see Section 2.1 and Section 2.2). Hence the boundaries of single systems are reducing in order to get advantage of new potential interactions with remote services or resources. Agents in open multi-agent systems are owned by a variety of stakeholders, with different aims. The environment changes dynamically and it is not possible to know a priori how it will change because agents act in an autonomous and flexible manner to achieve their goals, and they continuously and in an unpredictable manner join and leave the available platforms.

An open agent system cannot have a central authority that performs supervising tasks over the agent society, having a global perspective and a wide knowledge of the system. Furthermore each agent has a weak knowledge about the environment and about its peers. Complex networks of interacting autonomous agents without central control can be managed taking inspiration from social mechanisms to provide them with robustness and security.

Organizations and protocols cannot prevent agent systems from interaction risks that could threat security and reliability. Hence to reduce risks, trust concept is adopted from human society to agent society as a way to get information for facing the uncertainty related to agents' behavior. If we suppose that a typical distributed system based

on open multi-agent systems is composed of thousands of agents, security mechanisms that provide reliability in communication, confidentiality, integrity, authentication and non-repudiation cannot cover all emerging requirements.

The concept of trust that was described in Section 2.3 could play a central rule in the interactions among agents [47]. In closed systems a known agent can be a trustworthy agent, an unknown agent is always untrusted. In open systems the concept of trust measure gives more flexibility to interactions among agents. System security and openness can be balanced by adding trust information among actors that get involved in transactions. Trust plays a very important role as a mechanism of social control.

It is necessary for an agent to estimate the trustworthiness of each potential partner in order to decide whether the partner is reliable to interact with. The combination of trust opinions collected from entities that experienced interactions with the same target agent, produce trust from the social perspective, referred as reputation. Reputation is used for upsetting trust relationships among the underlying infrastructure of the agent platforms and the agents themselves. This structure, composed of trust relationships among entities, is called *network of trust* or *web of trust*.

3.2 Ontologies and conceptualizations

3.2.1 General features of ontologies

The term *ontology* comes from philosophy (*Aristotele*'s ontology), as a theory of the nature of existence: an ontology models a domain of knowledge. Philosophically we refer to an ontology as a particular system of categories accounting for a certain vision of the world. From the engineering perspective, an ontology is defined as a set of representational primitives with which to model a domain of knowledge or discourse [41]. It is an engineering artifact constituted by a specific vocabulary used to describe a certain reality, and by a set of explicit assumptions regarding the intended meaning of the words [43]. Classes or sets, attributes or properties, and relations among class members are typical representational primitives. Specific logical constraints are defined for consistency of their application. In the simplest case an ontology describes a hierarchy of concepts with specific relationships; in more sophisticated cases, suitable axioms are added in order to constrain the intended interpretation of the concepts.

Languages to define ontologies are semantic level languages with high abstraction and expressive power; ontologies are independent from the languages used to describe them (low level data models).

To better understand the meaning of ontology, it is necessary to introduce the *conceptualization* term: a conceptualization is a set of informal rules that constrain the structure of a piece of reality, which an entity uses in order to isolate and organize relevant objects and relevant relations [30].

A conceptualization accounts for the intended meanings of the terms used to denote the relevant relations. These meanings are supposed to remain the same if the actual extension of the relations change due to different states of affairs [42].

According to [44] an ontology is an explicit, partial account of a conceptualization. Referring to [40], ontologies are agreements about shared conceptualizations. Shared conceptualizations include conceptual frameworks for modelling domain knowledge, context-specific protocols for communication, and agreements about the representation of particular domain theories.

Hence a conceptualization is composed of the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them.
An ontology is defined as an explicit specification of a conceptualization. The ontology takes the form of the definitions in a representational vocabulary (classes, relations, and so forth) that provide meanings for the vocabulary and formal constraints on its coherent use. An ontology is a specification of an abstract data model (the domain conceptualization) that is independent of its particular form.

Ontologies can be distinguished on the granularity of details. An high detailed ontology requires the use of a rich language. A low detailed one, that uses shared inferences in mind for interpretation of part of the concepts of the underlying conceptualization, are easier to be used.

Another distinction can be made among an *ontology library* and an *application ontology*: the former contains more or less reusable knowledge across different applications, whereas the latter contains the definitions specific to the application [42].

Ontologies are used for interaction and for building knowledge base for problem-solving and decision-making. A typical problem in ontologies is the *interaction* problem, that consists in introducing a new concept in the application ontology, specific of a certain domain and a certain method, without any attempts to generalize it in order to reuse it in a more general task or domain.
Reusability should be systematically pursued: a model for a specific knowledge should be built using the most intrinsic and task-independent aspects of each piece of reality. Domain analysis is essential before the design of an ontology in knowledge-based systems: to organize a domain knowledge in a reusable and transparent way, it is necessary to find out the intrinsic structure of the domain knowledge.

W3C also defines ontology standards (the Resource Description Framework RDF and the Web Ontology Language OWL) for Semantic Web for data exchanging among systems providing standard conceptual vocabularies.

3.2.2 FIPA ontologies for agents

FIPA emphasizes the use of ontologies for communication among agents because communicative act theory, adopted by the agent model, requires the use of common ontologies to guarantee the consistency (common meanings of the expressions) and compatibility (common concept) of the exchanged information.

Agents do not use remote procedure calls (RPC) to get services from each others, they just exchange messages. The contents of the messages specify the requests. Furthermore agents do not need to know each other before starting a communication: they need a common language to understand each other. Hence agent communication requires a mechanism that support efficient and reliable expressiveness for message contents. For the above reasons, common ontologies are used by agents in their domain of discourse. The ontology ensures that the agents ascribe the same meaning to the content of the messages.

FIPA provides an *Ontology Service* (experimental) specification [30] that defines the role of the *ontology agent* (OA) in a platform which should discover public ontologies in order to access them, maintain a set of public ontologies, translate expressions between different ontologies and/or different content languages, respond to query for relationships between terms or between ontologies, and facilitate the identification of a shared ontology for communication between two agents.
The OA provides an agent community with ontologies that are stored in repositories. The OA supports agents who wish to start communication but that do not share a common ontology: for instance it could compare the ontologies that the two parties know in order to provide a common one.

FIPA also defines some ontologies to be used for basic service, e.g. the agent management ontology [31] employed for the description of the agent's identity by the directory facilitator (DF) and by the agent management system (AMS) services.

3.2.3 Ontologies and content management in JADE

The support for content languages and ontologies provided by the JADE platform [7], is designed to convert, at the sender side, the internal representation of a complex information contained in a message, into the corresponding ACL (Agent Communication Language) content expression representation, and to perform the opposite conversion at the receiver side. The receiver has to perform semantic checks to verify that the received information complies with the rules of the shared ontology.
Inside an agent, the information is represented as Java objects; the JADE support for content languages and ontologies (the *content manager*) converts this information in a string (or sequence of bytes) that is easy to transfer.

The content reference model, used to perform semantic checks, classifies the elements in the domain of discourse according to the ACL language defined by FIPA. At

the highest level it distinguishes *predicates* and *terms*:

- **predicates** are expressions that claim something that can be true or false.
- **terms** are expressions identifying abstract or concrete entities that exist. These are further classified into:
  - *primitives* are atomic entities (e.g. strings and integers)
  - *concepts* are entities with complex structure that can be defined as collection of slots
  - *agent actions* are special concepts that describe actions performed by agents
  - *aggregates* are entities formed by groups of other entities
  - *identifying referential expressions* are expressions that identify the entities for which a predicate is true
  - *variables* are expressions that identify unknown entities

An ontology for a given domain is a set of schemas defining the names and the structure of the predicates, the agent actions and the concepts. An ontology is an extension of the Ontology class to which schemas are added to define the types of predicates, agent actions and concepts relevant to the domain. Each schema included in an ontology is associated with a Java class that implements the Concept, or AgentAction, or Predicate interface. For each slot in the schemas, there must be a set and a get method.

The JADE package includes codecs for the SL, LEAP and XML content languages for content management. SL is a human-readable string-encoded content language. The LEAP content language is non-human-readable byte-encoded content language. The XML content language uses an XML syntax.

An agent has to register the defined ontology and the selected language to its content manager, which creates and manipulates content expressions as Java objects. The user works only with Java objects, but JADE uses abstract descriptor classes when translating content expressions. The conversion pipeline starts when the content manager fills or extracts the content of the message (Figure 3.1):

- the codec object associated to the selected language, converts s string or sequence of bytes into/from an abstract content element;
- the abstract content element is validated against its schema;
- the ontology object associated to the defined ontology, converts the abstract content element into/from a Java object of a class implementing the ContentElement interface.

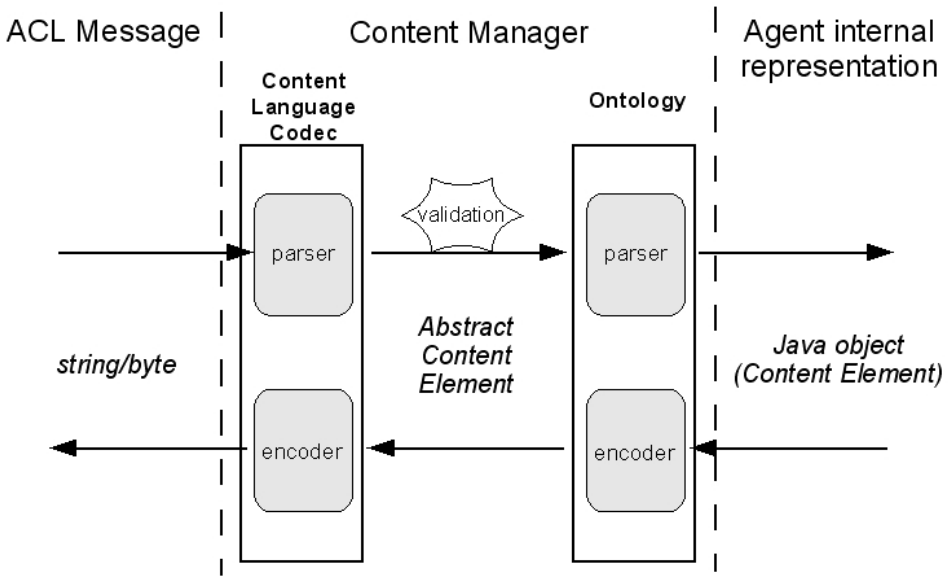

Figure 3.1: Conversion pipeline

The user is given the possibility of setting additional constraints called facets to the predicates, agent actions and concepts he/she defines in the ontology (e.g. checking if the integer that describes the cost of an item is positive).

An ontology allows a machine to handle the contents of messages. The user understands the meaning of the content, whereas the computer does not, in any deep sense, but it can manage the content through the defined ontology. Hence software agents can deal with the semantic of the user information.

3.3 Centralized reputation model for MAS

3.3.1 Weakness of actual reputation systems

Among primary challenges in MAS is the identification of suitable models and technologies going beyond the traditional concept of security, known as *hard* security that is based on identity authentication and authorization.
The notion of trust in artificial systems introduces new path for the development of alternative security mechanisms, called *soft* security, that are based on *trust evaluation* and *reputation management*. Suitable infrastructure are required to embed the notion of trust in order to effectively support expressiveness for this new abstraction.

The challenge is to model trust concepts coming from sociology, psychology, economy, law, in order to import it in MAS technology. The reputation model for agent systems that we propose is aimed at improving trust measures and the information it is required to provide, considering the specific features that characterize an agent systems

in open environment.

Actual reputation systems usually represent a trust network as a directed graph where edges represent weighted trust relationships (*sociogram* [71]). The direction of the edge defines the trust origin and the trust target; the weight expresses the degree of trust among two entities. Reputation systems uses the graph to discover paths of trust relationships: a path from the origin node to the target node describes a trust relationship among the two entities [51].

A sociogram uses the transitivity of the trust concept: if Alice trusts Bob and Bob trusts Claire, then Alice (indirectly) trusts Claire:

$$\left(\text{Alice} \overset{trust}{\rightarrow} \text{Bob}\right) \wedge \left(\text{Bob} \overset{trust}{\rightarrow} \text{Claire}\right) \Rightarrow \text{Alice} \overset{trust}{\rightarrow} \text{Claire}$$

Trust is not implicitly transitive. Once Bob has informed Alice with a recommendation that he trusts Claire, now Alice may or may not trust Claire (Figure 3.2). This depends on different factors, for instance what the degree of trust is from Alice to Bob, if she completely trusts Bob or not.

While trust is not inherently transitive, a form of transitivity can be achieved. For instance Alice may wish to combine all collected evidence, both her own observations and recommendations from other parties regarding the same subject, to form an overall trust in a trust target. Hence transitivity is in general subject of some conditions.

Trust could be a function of the length of the path (chain of trust) among origin and target in the sociogram, and a function of each single trust value encountered along the path.

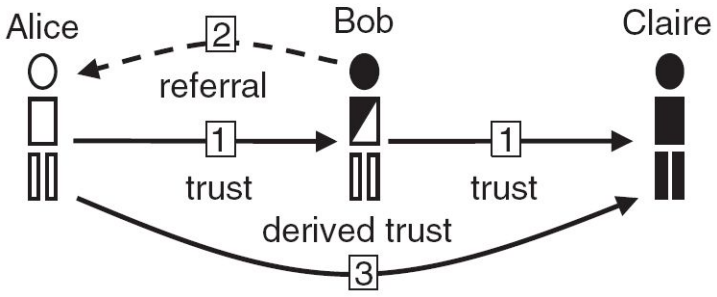

Figure 3.2: Trust transitivity

The management of a sociogram graph in dynamic environment could get challenging due to its increasing size: new entities imply new nodes; new trust relationships among entities produce new edges. Updating the graph of each entity is not trivial. In our opinion a reputation management system should reduce complexity in the process of information gathering related to entities, hence we think that the sociogram approach is not suitable for agent systems.

Several reputation models are based on a probabilistic perspective of trust information [52, 60, 17], for instance using Bayesian approaches for trust measurement and beta probabilistic density functions. Those models do not consider trust components that are relevant in trust opinion formation or reputation evaluations, e.g. beliefs, context characteristics and interaction risks.
Those models mainly focus on the computation of an overall trust value from numerical opinions collected by the community of participants; they do not give relevance to the opinion formation process.
The way an entity converts his beliefs in a numerical value that expresses the degree of trust is important for a reputation system, because it implies some issues related for instance to subjectivity of personal evaluations, that affect reliability of information. Trust opinions are affected by subjectivity if users do not have more specific parameters to base their opinions upon.

Actual reputation systems are affected by lack of personalization of trust values (as pointed out in [72]) resulting in coarse-grained trust and reputation value. Trust is a compound of beliefs, hence the simpler is the method used to express it, the higher is the risk for it to be not accurate and reliable. For instance a single numerical value that summarizes a degree of trust is far to be an accurate trust evaluation in a scenario of trust-based decision making.

Most of reputation systems collect trust information using general schemas for opinion formation. Such a general approach is suitable for all generic concepts of trust (e.g. commercial trust, delegation trust, social trust) but it does not consider relevant specifics that characterize the different contexts of trust. Information about the context where transactions take place should not be disregarded in the reputation model in order to improve the accuracy of the model. Opinions should also contain deeper and more detailed information about transaction dynamics in order to provide a more robust and reliable reputation model.

A reputation system that can provide users with accurate information is a consistent support for decision-making process related to the participation in transactions with not-well-known entities that are potentially risky.

3.3.2 Overview of the proposed model

The model we propose strongly couples the specific meaning of trust with the context it is related to, rather than using generic definitions of trust and reputation that are supposed to fit every context. From our perspective trust and reputation concepts are a compound of detailed and deeper features that improve the significance of the values that quantify them.

The proposed reputation model emphasizes the distinction among different contexts. Within each context trust and reputation information is independently managed,

allowing the system to deal more effectively with different subjective and specific features.
Our aim is to model trust and reputation concepts according to the specific context where they are employed in.

Ontologies are employed to collect relevant multi-dimensional information specific to each context. As ontologies provide a common language for agents of different systems [20], information generated can be shared among different contexts.

The architecture of a reputation system determines how opinions and reputation ratings are communicated among participants. There are two main types of architecture:

- centralized,
- distributed.

In a *centralized* approach, individual evaluations about agents' behaviors or capabilities are collected by a central service, which purpose is to provide an overall reputation measure for each participating agent. Reputation scores can be used by agents to decide whether or not to transact with a particular agents. The communication protocol is centralized: agents communicate with the central service. The reputation computation is performed by the central service using the received opinions.

In a *distributed* reputation system, there is no central location for submitting ratings or obtaining reputation scores. Information is spread among distributed stores or each participant simply records the opinion about each experience with other parties, and provides this information on request from other parties. Entities are required to gather reputation information searching different peers. Those peers form the trust information system. The communication protocol is distributed: agents communicate with each others. The reputation computation is performed by each agents using its own information and ratings received from the others.

Most of the reputation frameworks in the literature are based on a distributed management approach. Among these we can mention Abdul-Rahman and Hailes [3], SECURE [19], Yu and Singh model [84], PeerTrust [83], Buchegger's system [16], and ReGreT system [70].
In the next section we will describe the centralized approach that we adopted.

As we described in Section 2.3, trust is based either on direct or on indirect experiences. In the proposed model, reputation is calculated using only opinions of direct experiences.

The model we developed separately considers two concepts:

- *reputation*,
- and *trustworthiness*.

The concept of reputation is strictly coupled with the context: it expresses the degree of reliability of an agent regarding the subject of the context group, its dependability related to the content and performance of the transactions which are evaluated by the context group.
We call reputation *context reputation*.

The concept of trustworthiness expresses the general honesty of an agent and is not related only to the specific context, but it has a wider meaning.
For instance an agent could have a good reputation related to the way it performs a service, but from the social perspective its trustworthiness can be weak due to unfair or illegal behavior. We call it *social trustworthiness*.

Agents can provide two kinds of opinions regarding the experienced behavior of another agent:

- the *context opinion* that rates the reliability of the performed service or actions,
- the *trust opinion* that expresses the general honesty and trustworthiness of an agent.

3.3.3 Centralized architecture approach

The reputation framework we developed, uses a centralized architecture approach (Figure 3.3). The framework is composed by independent groups (the *context groups*) of agents, characterized by a common domain or context of interest that binds all the participants, as also described in [9]. Trust evaluations and reputation scores are all related to the specific context the set was formed upon. A context group manages all the information relevant to reputation formation.

Agents join a context group because they are interested in getting trust scores collected by other members, relative to a common domain of interest. Members of the context group are remote or local agents. For instance, in an online auction service, seller and buyer agents rank their reciprocal reliability: grouping together in a context group they can collect and retrieve reputation information useful to decide if accepting or not transaction requests with not well-known entities.

A context group represents a coalition of entities interested on a specific topic hence collected information comes from entities that are likely specialized in that topic. Hence agents that are retrieving information can collect likely reliable information easier than requesting it to several other agents with an uncertain successful probability. Furthermore, a context group allows agents that did not know each other previously to get in touch and to share information that can be helpful to both of them.

The principal of each group is the *context manager*, an agent (services in MAS are provided by agents) permanently hosted in a platform, whose task is to collect opinions produced by members and specifically related to the context. The manager forms

reputation scores using those gathered evaluations (opinions). It distributes reputation rates to members of the its group, providing information about reliability of agents. Information are exchanged more efficiently in a group that is structured as a federation rather than in an unstructured set of agents: information is retrieved through the central manager rather than through a single replicated requests.

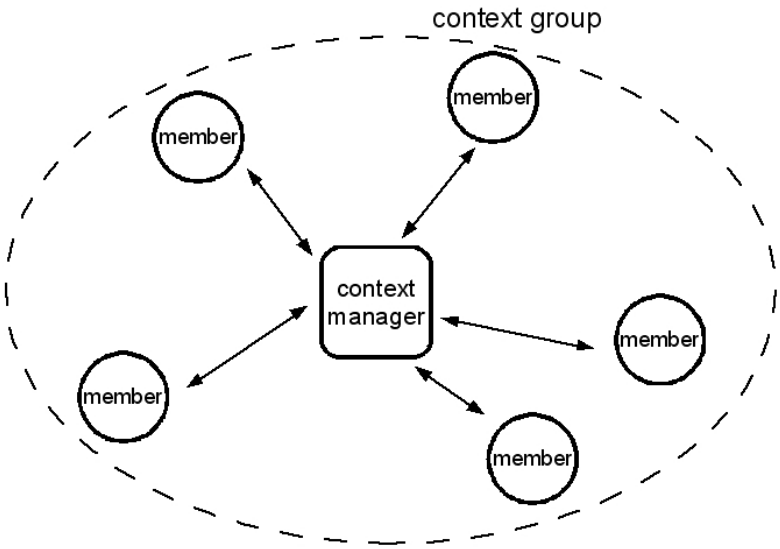

Figure 3.3: Centralized context-dependent reputation framework

Our model uses a centralized architecture, because we can benefit of the following advantages:

- information is completely stored in the central context service; agents are not required to spread requests of information to their peers; information location is not uncertain
- members do not need to get involved in reputation maintenance (as described in [20]) requiring them to inform other members with updates of their own opinions; information they get from the manager is the updated if we assume that agents are willing to share their experiences;
- no risk of late incoming replies to queries (due to network or request handling latency, caused for instance by a long path of entities that the query has been forwarded to), because queries are directly handled by the manager;
- user applications are decoupled from the reputation system, in fact agents do not need to embed mechanisms for trust information management; for instance members do not have to combine opinions to obtain a reputation value.

The centralized approach fits better the nature of MAS: communication is often a weak point in agent systems because software agents are applied in mobile devices,

for instance in pervasive and ubiquitous computing, where resources are limited, especially bandwidth is weak and network connection is unstable. This approach allows the system to provide information whether members are offline, because queries are not sent to agents, as for the distributed approach, but agents send requests to the group manager.

Agents can join more than one group, according to the available context groups and to its interest. This does not imply an increase in its computational load but it just allows it to exchange trust and reputation information more contexts. In Figure 3.4 we show an example of agents running on different platforms that joined several context groups: for instance agent1 joins group B, C and D.

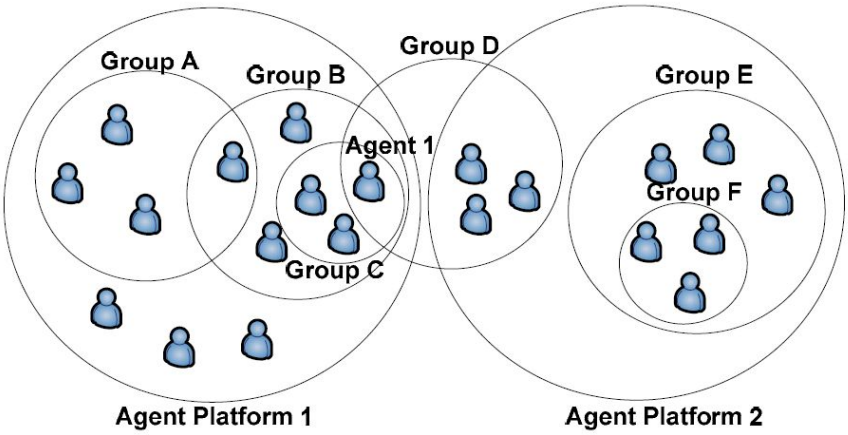

Figure 3.4: An example of agents that join several context groups.

A centralized architecture is affected by reliability issues: the reputation service strictly depends on the effectiveness of the manager agent. This issue in MAS is typically faced with service replication in order to enhance reputation system reliability: agents that provide services can be cloned in the same host or in another one in order to guarantee the service in a seamless manner in the case that an agent crashed.

Within this approach reputation becomes a group concept. When the manager evaluates that user A has a *'good'* reputation rate, it means that the group trusts A as a good partner, but it does not mean that each single member trusts A as a good partner. Although the concept of trustworthiness is extended to the group, this approach reduces subjectivity in trust evaluation.

Grouping agents in contexts of interests is also useful because agents joining a specific group are supposed to be familiar with the context, hence opinions are likely to be reliable because they come from experienced users. A context group is supposed to be a coalition of agents that have deep knowledge of the matter that is related to the context.

3.3.4 Reputation ontologies

Opinions from agents should be a compound of detailed and specific evaluations of different aspects of the behavior of agents that participated in interactions.

We apply ontologies for trust evaluation as a promising approach to solve the lack of expressiveness of reputation measures, and highlight contextually relevant information in the reputation measures [36].

According to ACL language (Agent Communication Language) specifications of *FIPA* (the Foundation for Intelligent Physical Agents), an ontology defines the proper semantic of the message content and its performative (Section 1.6).

The ontology provides the schema for opinion formation: agents are required to provide an evaluation for each parameters contained in the schema. The parameters emphasize specific characteristics of the context that are relevant for an accurate evaluation of the agents' behaviors.
Once an opinion is sent to the central manager, the parameters that it contains are used for the computation of the context opinion, that is an overall rating of the reliability of the agent.

Reputation systems usually handle opinions that are single numerical values that evaluate the overall quality of the interactions. An agent could take into more consideration an aspect of the interaction and another agent could focus more on another aspect. Hence the opinions can be very different even if the quality is the same. Splitting the parameters of evaluation that an agent is required to rate helps the formation of a more objective opinion. Subjectivity in opinion formation is reduced when specific aspects of an agent's reliability are evaluated. Hence the collected information, that is related to experienced interactions among agents, is more objective.

Two ontologies are employed. The first is a generic ontology, the *coarse-grained ontology* (Table 3.1), that is used by all context groups for basic communication among members and their manager. This ontology contains concepts that are common to all the groups:

- `<Entity>` contains the description that identify an agent of the group
- `<Context>` contains the description of the context group
- `<ContextReputation>` is the quantification of an agent's reputation computed by the group manager
- `<SocialTrustworthiness>` is the quantification of an agent's trustworthiness computed by the group manager

The `Trust` predicate is used by an agent member to send a trustworthiness opinion about another agent to the manager.
An agent in the group sends a query performative message (a query according to the

#	TYPE	NAME
1	concept	**Entity**
2	concept	**Context**
3	concept	**SocialTrustworthiness**
4	concept	**ContextReputation**
5	predicate	⟨*EntityX*⟩ **ParticipateTo** ⟨*this group*⟩
6	predicate	⟨*EntityX*⟩ **IsReputated** ⟨*ContextReputation*⟩
7	predicate	⟨*EntityX*⟩ **IsTrusted** ⟨*SocialTrustworthiness*⟩
8	predicate	*I* **Trust** ⟨*EntityX*⟩ as ⟨*SocialTrustworthiness*⟩
9	action	**Insert** ⟨*EntityX*⟩

Table 3.1: Generic reputation ontology

FIPA ACL specifications described in Section 1.6) to the manager to know if the mentioned entity is a member of the context group, hence it is reputated: `"<EntityX> ParticipateTo"`. If the manager finds information about the entity, it sends an informative message to the requester `"<EntityX> IsReputated <Reputation>"`.
The `IsTrusted` predicate is used by the manager to give information about an agent's social trustworthiness rating.
The `Insert` is used by an agent in a request performative message for the manager: it requests to add a trusted agent as a new member of the group.

Figure 3.5 sketches out some example of requests and replies that occur between the manager and a member. A request has the respective reply on the same level; different requests stand on different levels, represented by dotted lines.

The second ontology is specific to the context group. We call it the *fine-grained ontology*. It is developed and provided by the group activator: he is required to define the parameters that it contains.
Hence, this ontology schema is used by the agents in the group for communication of context opinions to the manager. The context opinions contain specific information related to the ability and dependability of agents regarding the specific context. The manager uses the context opinion to calculate a single numerical value that is added to the other context opinions that evaluate a given entity.

An example is provided in Table 3.2: it regards a context group where members are investors that evaluate stockbrokers. Investors communicate opinions related to the reliability of the broker, the quality of traded shares, the confidence in the broker and the broker's readiness to market changes. Values range from 0 to 1 for positive opinions, from 0 to -1 for negative ones.

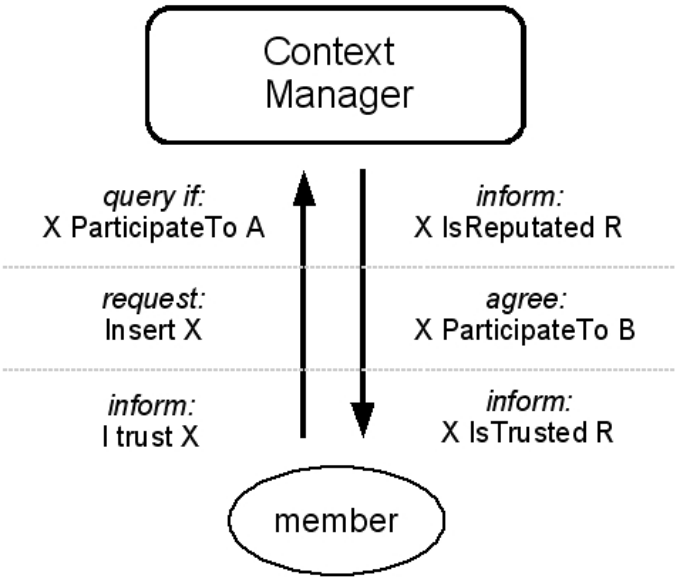

Figure 3.5: Examples of requests and replies between a member and the manager

#	Type	Evaluated Parameter
1	concept	**BrokerReliability**
2	concept	**ShareQuality**
3	concept	**Confidence**
4	concept	**Readiness**

Table 3.2: Example of context-related ontology

3.3.5 Reputation and trustworthiness computation

The manager computes context reputation and social trustworthiness values of the participants of the context group.
The manager stores context and trustworthiness opinions in two matrixes R and T. The columns of the matrixes are the rated agents, the rows are the rating agents.

In the matrix T, $t_{i,j}$ is the last trustworthiness opinion that agent i gave about agent j. A trustworthiness opinion is intended to rate the honesty and reliability of an agent in a general perspective. $t_{i,j} \in [-10, 10]$: if $t > 0$ it means that the agent is trusty, if $t < 0$ it is not trusty.

In the matrix R, $r_{i,j}$ is the last context opinion that agent i gave about agent j. A context opinion rates the dependability of the agent related to the specific context. $r_{i,j} \in [-10, 10]$: if $r > 0$ it means that the agent is reliable regarding the topic of the context, if $r < 0$ it is not reliable.

The proposed reputation model store just the last context opinion $r_{i,j}$ and the last trustworthiness opinion $t_{i,j}$ from agent i regarding agent j. The manager computes an agent's context reputation from the R matrix, and its social trustworthiness from the T matrix.

The model does not store past opinions from agent because the last opinion is the most relevant for the sake of reputation and trustworthiness evaluation. This provide a more flexible rating that can dynamically and quickly face changing situations: for instance when an agent changes its behavior from honest to dishonest it is necessary to consider actual behavior and new opinions, disregarding past positive opinions. Past opinions produce a kind of inertia for reputation and trustworthiness values to change according to the most actual opinions. Our model is not aimed to predict the future behavior of an agent considering its past behavior. It is supposed to evaluate the actual agent's behavior according to the last experienced interactions that other agents had with it.

The social trustworthiness Tw_j of the agent i represents the honesty of the agent, hence it used as a degree of the reliability of the opinions it provides. Tw_j is the weighted average of all the trustworthiness opinions, both positive and negative, provided by agents whose social trustworthiness is greater than a threshold $\overline{Tw} \geq 0$ that was fixed at the creation of the group:

$$Tw_j = \frac{\sum_i tw_{i,j} Tw_i}{\sum_i Tw_i} \ , \quad Tw_i > \overline{Tw}$$

This is the *social trustworthiness* value of the member j in the specific context group.

Two distinct overall rates are computed, both for the context reputation and for the social trustworthiness:

- the positive rates Rep_j^+ and Tw_j^+ for agent j, computed with the positive opinions
- the negative rates Rep_j^- and Tw_j^-, computed with the negative opinions.

Rates are computed with the weighted average of the opinions. The weight is the social trustworthiness Tw_i^+ of the agent that provided the opinion. Only opinions that belong to positively trusted ($Tw_i > 0$) members are used for context reputation and social trustworthiness formation.

The two social trustworthiness rates of agent j are:

$$Tw_j^+ = \frac{\sum_i t_{i,j} Tw_i}{\sum_i Tw_i} \ , \quad t_{i,j} > 0$$

$$Tw_j^- = \frac{\sum_i tw_{i,j} Tw_i}{\sum_i Tw_i} \ , \quad t_{i,j} < 0$$

The two context reputation rates are:

$$Rep_j^+ = \frac{\sum_i r_{i,j} Tw_i}{\sum_i Tw_i} \ , \quad r_{i,j} > 0$$

$$Rep_j^- = \frac{\sum_i r_{i,j} Tw_i}{\sum_i Tw_i} \ , \quad r_{i,j} < 0$$

The context reputation and the social trustworthiness values provided by the manager contain compound information (Table 3.3 and Table 3.4) that is useful to better understand how the group rated an agent. This information is aimed at giving a complete and clear view of the evaluation that the agents in the group provided. The better is the information provided the better is the support for decision making.

The context reputation contains: the positive and the negative context reputations (Rep_j^+ and Rep_j^-), the number of the positive opinions and the negative opinions that contributed to both the reputations, and the average of the social trustworthiness values of the agents that provided the positive and the negative opinions.

The social trustworthiness contains: the positive and the negative social truwtworthiness values (Tw_j^+ and Tw_j^-), the number of the positive opinions and the negative opinions that contributed to both the values, and the average of the social trustworthiness values of the agents that provided the opinions.

CONTEXT REPUTATION	
+	–
+ reputation	– reputation
# of + opinions	# of – opinions
avg trustworthiness of + raters	avg trustworthiness of – raters

Table 3.3: Context reputation value provided by the manager

SOCIAL TRUSTWORTHINESS +	SOCIAL TRUSTWORTHINESS −
+ trustworthiness	– trustworthiness
# of + opinions	# of – opinions
avg trustworthiness of + raters	avg trustworthiness of – raters

Table 3.4: Social trustworthiness value provided by the manager

The following example shows how the social trustworthiness is updated when the manager receives a trustworthiness opinion. It shows also an example of the format used for the social trustworthiness and context reputation information.
Let us suppose that the context group is composed of six agents, and that the trustworthiness opinions in matrix T for agent 4 are:

$$T = \begin{pmatrix} \cdot & \cdot & \cdot & 8 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 5 & \cdot & \cdot \\ \cdot & \cdot & \cdot & -2 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \epsilon & \cdot & \cdot \\ \cdot & \cdot & \cdot & \epsilon & \cdot & \cdot \\ \cdot & \cdot & \cdot & 7 & \cdot & \cdot \end{pmatrix}$$

An ϵ entry in the matrix means that no opinions are available.
Let us assume that:

- the actual social trustworthiness values of the agents are $Tw = (7, 4, -3, 6, 8, -2)$,
- $\overline{Tw} = 2$,
- agent 5 sends a new trustworthiness opinion: $t_{5,4} = 7$,

The manager computes the weighted average for the new social trustworthiness value for agent 4 according to the new opinion: $Tw_4 = 6.5$.
$t_{3,4}$ and $t_{6,4}$ are not considered in the computation because $Tw_3 < \overline{Tw}$ and $Tw_6 < \overline{Tw}$.
The new social trustworthiness values are $Tw = (7, 4, -3, 6.5, 8, -2)$.

The social trustworthiness information regarding agent 4 that the manager provides is showed in Table 3.5.

SOCIAL TRUSTWORTHINESS	
+	−
6.5	ϵ
3	0
6.3	ϵ

Table 3.5: Social trustworthiness information example.

$t_{3,4}$ and $t_{6,4}$ are not considered because $Tw_3 < 0$ and $Tw_6 < 0$.

Let us suppose that the context reputation matrix column for agent 4 is:

$$R = \begin{pmatrix} \cdot & \cdot & \cdot & 6 & \cdot & \cdot \\ \cdot & \cdot & \cdot & -3 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 4 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \epsilon & \cdot & \cdot \\ \cdot & \cdot & \cdot & -2 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 5 & \cdot & \cdot \end{pmatrix}$$

Hence the context reputation information regarding agent 4 that the manager provides is showed in Table 3.6.

CONTEXT REPUTATION	
+	−
6	-2.3
1	2
7	6

Table 3.6: Context reputation information example.

$r_{3,4}$ and $r_{6,4}$ are not considered because $Tw_3 < 0$ and $Tw_6 < 0$.

3.3.6 Other features of the model

In the following we describe details and features of the reputation management model.

Startup: a context group is created by a request of a user to the agent platform in order to activate a reputation manager with the description of the context that it is going to manage. The manager supports services in its context group (e.g. membership management, opinions collection, reputation and trustworthiness computation, information request handling). The user must provide both the context specific ontology

for opinion formation, and the weights that are used for computing a single numerical value (that is stored as a context opinion) from the evaluated parameters contained in the opinion messages sent by agents, according to the specific ontology schema.

Memberships: an entity enters a group when it is interested in getting reputations about the behaviors of other entities belonging to the group. Groups are open to external agents who are trusted by a member and that can get a positive membership recommendation.
Agent platforms provide users with the Directory Facilitator service (*DF*) to retrieve the list of active context reputation groups in the platform.
The recommended entity joins the group with an initial social trustworthiness Tw value that is a third of the recommender's value. This proportion is heuristically chosen. The assignment of low trustworthiness value to newcomers is an effective mechanism for discouraging participants to misbehave and subsequently change their identity.
Only members with positive reputations can recommend for membership. The group activator fixes a minimum positive reputation value ($\overline{Tw}_{rec}$) as threshold for requesting recommendations to the manager. This mechanism avoids untrusted entities joining the group, and reduces the capabilities of weakly trusted agents to recommend new participants. An external entity that cannot get at least one positive recommendation is supposed to be potentially malicious, hence it could provide unfair opinions in the group, corrupting entities' reputations.

Member's proxy: if the creator of the group requires authenticated communication among agents and manager, once an entity joins the group, a proxy is created in the entity's host (or, if not allowed, in the manager's one): the proxy receives messages (requests and opinion) from the agent, authenticates and forward them to the manager (Figure 3.6). Hence communication from the proxy to the manager is secure. The authentication process does not involve the agent because it is performed by the proxy.

Opinion formation: a member sends a context opinion about another entity to the manager using the specific ontology schema (fine-grained ontology) of the context group. Opinions provide evaluations of the agent's beliefs regarding aspects of a typical interaction related to the context that occurred with another agent. Each aspect is evaluated in the range $[-10, 10]$. The manager computes a weighted average of the values contained in the opinion, producing an overall context opinion value.
Agents provide trustworthiness opinions using the generic ontology. Those opinions range between -10 and 10.

Once the manager receives an opinion, it updates the context reputation or the social trustworthiness of the related agent (Figure 3.7).

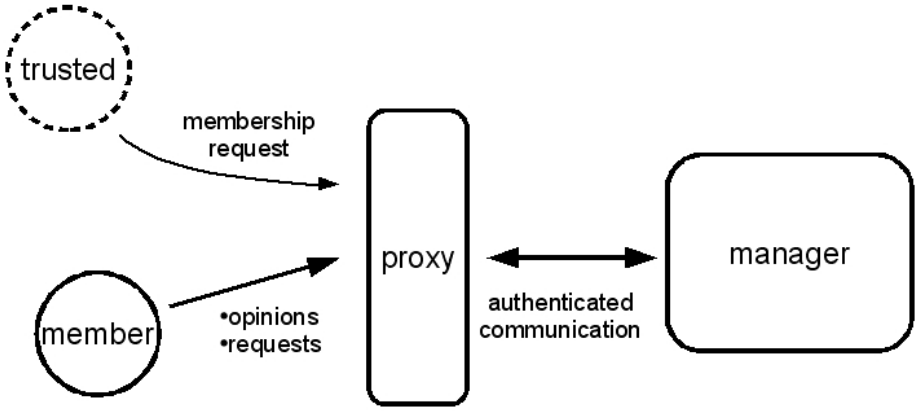

Figure 3.6: Proxy for communication between member and manager

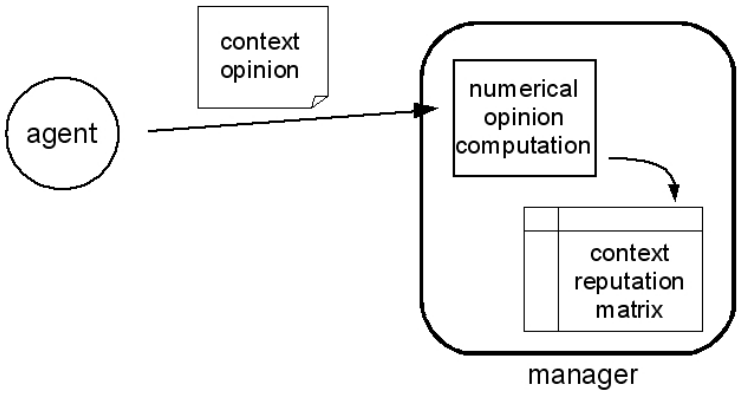

Figure 3.7: Opinion evaluation

3.3.7 Scalability

In open agent systems scalability is a major issue. Our model was aimed to be fully scalable to meet low-computational load requirements. A reputation management system should not affect the overall performance of the system because its purpose is just informative.

In decentralized reputation systems scalability is not a trivial issue because the bigger is the size of the web of trust, the harder is to know where to retrieve reputation information. This results in high bandwidth consumption for query spreading: an agent sends an entity's reputation query to the agents it trusts, and if those agents do not have the requested information, they should forward the query. For unknown agents this implies that the forward process can indefinitely continue, unless some limiting mechanisms have been defined (for instance the maximum number of hops). The number of spread queries can be proportional to the size of the agent community.

Information queries could also affect the performance of an agent that could get too busy with query management.

Our positive contributions to scalability of the reputation system come from splitting the overall community of agents in reduced size context groups to be managed.

The proposed model provides context managers with the ability to start new managers of the same context but in different hosts. The new peer manager will take care of future new members of the group. *Peer* managers can cooperate to serve information requests (Figure 3.8). If a manager does not have information about a requested entity, it can forward the information request to peer managers of the same context.

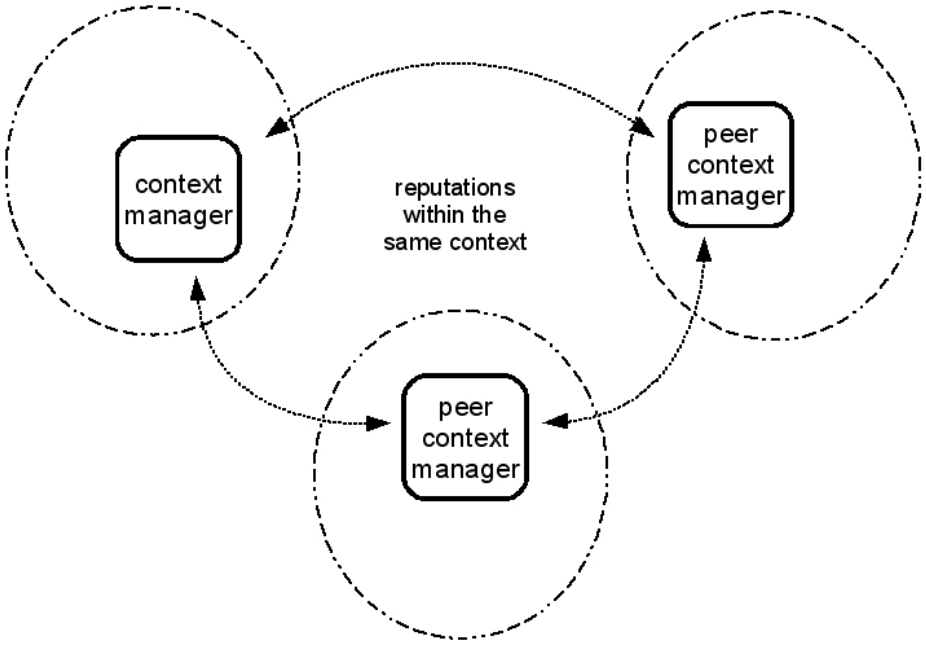

Figure 3.8: Peer managers of the same context group

3.3.8 Unfair rating

The quality of a reputation system depends on the integrity of the opinions it receives from the participants. Agents can rate more positively or more negatively than the real experience would dictate. For instance buyers could intentionally provide unfair positive opinions or unfair negative opinions for sellers. In large scale, open, dynamic distributed environments agents may possibly be self-interested, giving unfair rating for their own benefit, at the cost of others. Unfair rating can consist of a number of colluding agents which cause flooding of unfair rates. It can be used to manipulate the reputation system to an agent's advantage.

A fundamental issue in reputation systems is to avoid or reduce the influence of

unfair opinions. Effective protection against unfair ratings is a basic requirement in order for a reputation system to be robust. Usually unfair rating problem is faced with statistical filtering techniques ([85, 25, 5]), assuming that unfair ratings can be recognised by their statistical properties.

The development of a mechanism that checks the honesty of group members and their opinion integrity requires a very deep analysis, and it is out of the scope of this thesis. Only a trusted authority would be entitled to claim that a members is unfair in order to take advantage of their ratings. In an open agent system where all entities are peers it is not possible to entitle an entity to be an authority trusted by everyone. Furthermore agents should be closely controled by that authority.

To perform such a supervisory control we believe that it should be necessary to introduce an infrastructure based on sociological organization theory and social science concepts [63].

In [85] they introduce a context-based approach to filter out unfair recommendations using incremental learning algorithm. The context is used to analyze the user's activity, state and intention. The learning of context is incrementally increased by a neural network.

In [25] they try to reduce the effects of unfair positive rating using collaborative filtering techniques: if x provides an opinion about y, they determine a x's neighborhood set of raters whose ratings over many subjects are similar. In that group of raters they apply a cluster filtering algorithm in order to separate the set of raters into two clusters, that of fair and that of unfair raters. The final y's reputation estimate is then calculated with the ratings coming from the set of fair raters. They assume that fair and unfair ratings follow the same distribution type but with different parameters.

In [5] they propose a filtering technique that applies to both unfairly positive and unfairly negative ratings in Bayesian reputation systems. They assume that ratings provided by different raters on a given agent will follow more or less the same probability distribution. Hence when an agent changes its behavior all honest raters should change their ratings accordingly. The Bayesian system computes reputation scores by statistical updating of a beta probability density function (PDF). The reputation score can be represented in the form of the beta PDF parameters that represent the amount of positive and negative ratings. Their filtering algorithm is based on the beta distribution: a rater is considered unfair and its ratings are excluded when its score about an agent falls out of the lower q quantile and the upper $(1 - q)$ quantile of the beta distribution.

The above mentioned techniques exclude rates that are different from those of the majority of the raters.
We think that this can be effective if the number of raters is high enough that the majority of them can be considered honestly behaving, fairly rating. If there are few raters, most of them could be unfair, hence the above techniques can not face the effects

of unfair ratings.
We also think that those techniques slow down how quickly the reputation system can face changing agents' behaviors: if a well reputated seller becomes dishonest, the former agents that realize this change and that provide negative scores for that seller, are excluded from the computation of its reputation, hence the seller's reputation remains positive.

From a general perspective, opinions are subjective, and transaction experiences among agents differ each others: two opposite ratings regarding the same agent are not required to be necessarily one fair and the other unfair.

Our model detects potential unfair members relying on the trustworthiness opinions provided by the group. We assume that the majority of the agent in a group are fair and honest. Our approach is based on:

- reducing the possibility that a dishonest agent has to be included in the reputation computations
- providing detailed and complete reputation and trustworthiness information in order to support agent's decision making process with evident information.

Agents can join context groups only through recommendations of agents that are members of the group whose social trustworthiness is greater than a threshold, hence if we suppose that members with positive trustworthiness are honest, chances for malicious entities to join a group are very low.

An agent's opinion is included in the computation of reputations only if its trustworthiness is positive ($Tw_i > 0$), and it is weighted according to that trust degree. The assumption is that raters with low trustworthiness are likely to give unfair ratings and vice versa. This reduces the effects of ratings from agents that are not completely trusted from the group.

The context reputation and social trustworthiness values, that are provided by the manager, separate positive and negative contributions (Table 3.3 and Table 3.4). This format highlights if the opinions that form the reputation either clash because opposite, or are homogeneous. This allows the agent to more accurately decide to start a new transaction with the rated party. Decisions depend on the involved risk that is felt by the user too.

In [52] the reputation score of the multinomial system can be represented on different forms (evidence representation, density representation, multinomial probability representation, point estimate representation). While informative, the former three of them require not trivial computation to be displayed.

The reputation score representation of the proposed model, that consists in a vector of six numerical values, is both informative and easy to handle. Figure 3.9 shows an example of a context reputation value composed of both positive and negative opinions:

the heights of the two bold lines represent the positive and negative reputation values, and their position in the xy plane represents the number of raters and their average social trustworthiness.
The positive reputation value is 8 over 10, calculated over 60 opinions with an average trustworthiness of 7 over 10. The negative reputation value is 3 over 10, calculated over 20 opinions with an average trustworthiness of 3 over 10.

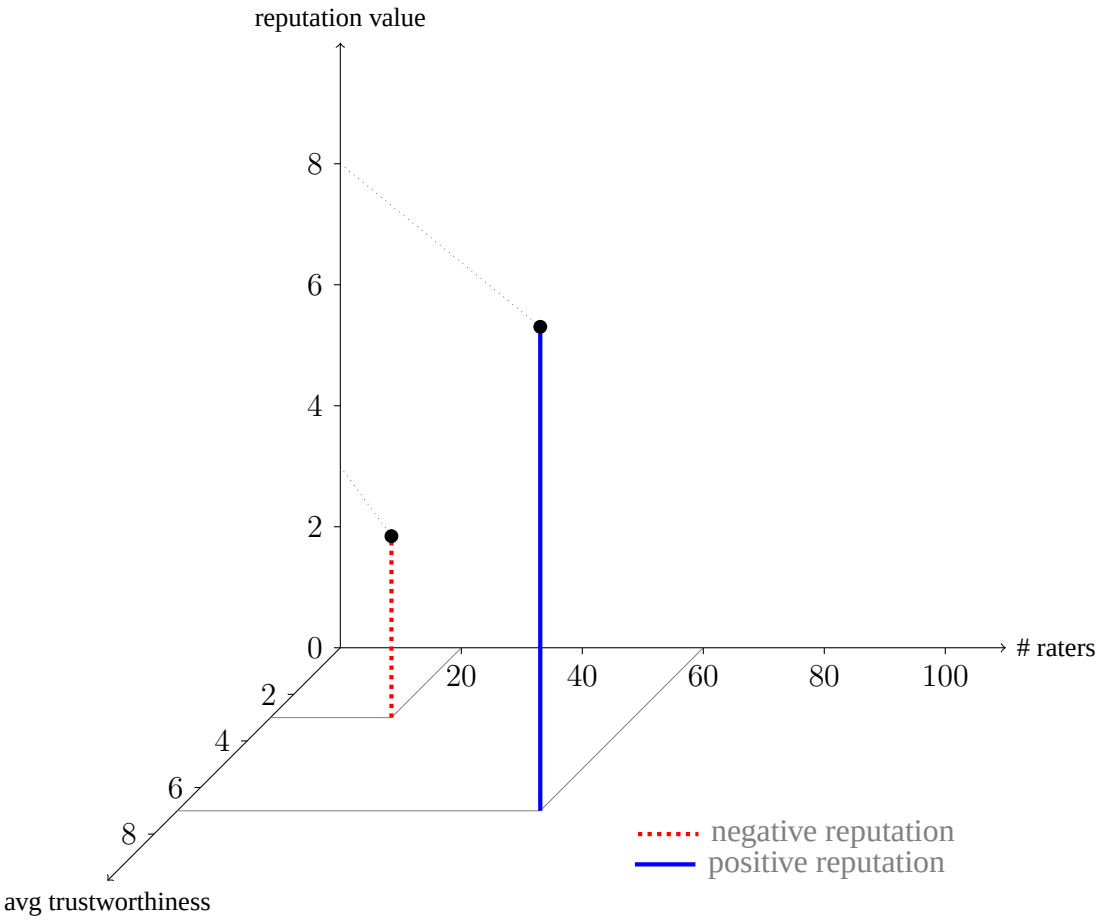

Figure 3.9: Graphical representation of a reputation value

This format provides a light but clear representation of how the community rated an entity. For instance this representation highlights how rated the majority of the group, and how rated the most trusted agents in the group.

3.3.9 Time

Reputation systems use time element to reduce the weight of the opinions as they get old (aging rates). The proposed model does not apply time decay to opinions because:

- once an agent provides an opinion, its value for the reputation computation should remain unchanged, otherwise it should periodically refresh opinions, and

there is no reason for it to do that;

- if opinions decay with time, malicious raters could frequently refresh their unfair rates in order to temporarily affect reputations.

Our model maintains the value of each opinion. The opinion of agent x about y is considered stable until x provides a new opinion and the previous one is removed. Nevertheless, the creator of a group is required to set up the duration of an opinion before it is definitively removed. This avoids opinions to be included for ever in the reputation calculation. The duration depends on the dynamism of the context, for instance the opinion duration of a car selling context is supposed to be longer than that of a context related to wineries, because purchase of wine is more frequent than that of cars.

Chapter 4

Social infrastructure based on the reputation system

4.1 Agent society and organizational paradigms

An agent system cannot be reduced to a compound of autonomous interacting entities. A system of agents gives rise to a society of agents, hence design of multi agent systems involves sociological organization theory.

A society is a collection of different entities grouped together with enduring and relatively stable patterns of relationships. A social structure may be seen as a set of organizational services, patterns, rules that underlies a social system and that partially guides actions of agents. From the social perspective, agent systems in open environment are collections of independent components that generally do not know each other. An effective multi-agent system should provide agents with means to efficiently cooperate and collaborate. The environment where they act should implement rules that drive the evolution of the system, and agents should incorporate different roles in that environment.

Sociality and autonomy key features of software agents would be suitably applied to open systems once we provide structures to improve integration among service provider and requester agents. The concept of social intelligence introduced in [87], highlight the intelligence ascribed to agents as a society, not individuals. This requires the definition of suitable social abstractions that incorporate organizational structures that govern and support agent interactions.

Mobile agents paradigm is well suited for the development of open systems, where agents from diverse remote locations, with different objectives, unknown to each other at design time, may interact to achieve their goals. Open systems can be conceived as a system as a whole that requires the involvement of sociological structures and mechanisms for the organization and management of the composing entities [63].

The distributed nature of MAS turns out in a not trivial complexity for the design

of these systems. Open systems increases even more that complexity because:

- it implies a dynamic environment;
- components are not known in advance and change dynamically;
- resources change their availability very frequently;

Openness imposes to consider the expected behaviors of agents and requires an overall system protection from misbehavior of agents. As a social abstraction, dealing with cooperation and coordination requires normative structures supported by organizations and institutions [74]. Social order is the multi-agent system equivalent of robustness in classical social engineering.

Agent-centered perspectives used for the development of MAS, have to be replaced with methodologies that employ organizational approaches (Gaia [86], Tropos [14], SADDE [75]) with an extensive use of the concept of agent's role, role dependencies, interaction protocols.

The absence of a hierarchy in open environments implies that the relationships in agent systems are close to relationships among peers. To provide a structure to the system, some researchers propose a form of democracy to be implemented [62] in the agent society. Democracy models such as the wise elite democracy, the rational choice model or the deliberative one, are taken into consideration at design time of a possible organization of an agent society.

Many researches have demonstrated that the organizational design employed by an agent system can significantly effects its performance [46]. Organizational paradigms that are used in multi-agent systems are:

- hierarchies
- holarchies
- coalitions
- congregations
- teams
- societies
- federations
- markets
- matrix organizations

Those approaches have different features which may be more suitable for some problems and less for others. The organization of an agent system consists of collections of roles, relationships and structures. The organizational structure supports agents in their interactions, hence the better is organized the community of agents, the more complex are the behaviors they can perform.

The context groups that are applied in the proposed reputation management model, can be described as coalitions of agents. The inner structure of a coalition is typically flat (peer agents), although we defined the manager agent that acts as intermediary and representative of the group as a whole. As described in [46], the general motivation behind the coalition formation is that some utilities can be gained by working in group.

4.2 Reputation-based social infrastructure

Multi-agent programing paradigm can be extensively employed in the development of distributed computing systems where independent components are structured according to social related concepts implemented in an infrastructure that provide an overall organization. Mobile agent system designers usually recognize the utility of a social structure to be applied to the community of agents.

Patterns of actions can emerge from interactions among agents. This turns out in the identification and definition of roles and rules that can simplify the interactions. In a complex agent-oriented system, those roles and rules are a partial solution. If agents are considered as individually acting entities in wide distributed systems, they have limited chances to know all the resources that the system could provide to support their tasks. If agents are collections of units that are parts of a structured system, the complexity and the size of the system are hidden by the structure that leads the agents in resource discovery.

A social structure benefits a community of agents because a component of the system can find out another one to interact with through a path of knowledge that is provided by the organized structure, hence the agent system gets a more efficient resource management.
Organizational concepts become key features for building large scale and complex systems [28].

We propose a social infrastructure based on our reputation model that is aimed at supporting reputation information flow in the society of agents. The main component of the proposed infrastructure is the *informative point*: it is a platform resident service that collects and spreads information of existing context groups and their belonging members.

The informative points are aimed at spreading knowledge about competence and integrity of individuals and service providers in the system. Each agent platform is supposed to have an informative point, but this is not mandatory because it is possible

to interact with informative points that are located in remote platforms.
Agent platforms implement directory facilitator (DF) services (yellow pages) that can provide information regarding the available informative points located in the platform. Informative points are required to subscribe to the DF service.

A network of informative points that is distributed over the system (Figure 4.1) connects parts of the agent society providing channels for reputation information flow. The wider is the network, the easier is to get information regarding agents. This benefits the chances that an agent has to interact with unknown agents.

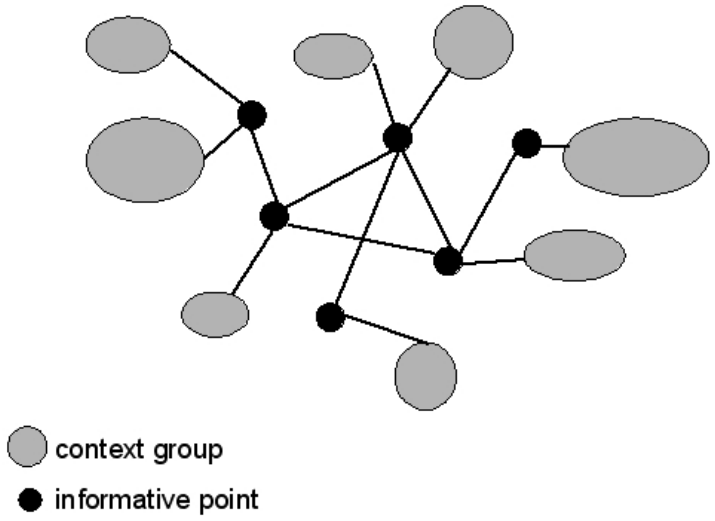

Figure 4.1: Social structure of networked informative points

Requests to an informative point could come from (Figure 4.2):

- agents that require information about other agents or information regarding existing context groups that are linked to this point;
- context managers that request trustworthiness information related to specific agents.

At creation stage, the context manager of a context group is required to register with the list of an informative point in order to create a bridge among the group and the society of agents. During its life, the context manager informs the informative point about the agents' identities that are registered with its group.

Each informative point manages the list of the context managers that are registered with it: it is used to give information to requesting agents about the known context groups.
It also manages a list of agents' identities that are registered with the above mentioned context groups: when the informative point receives an agent information request, it

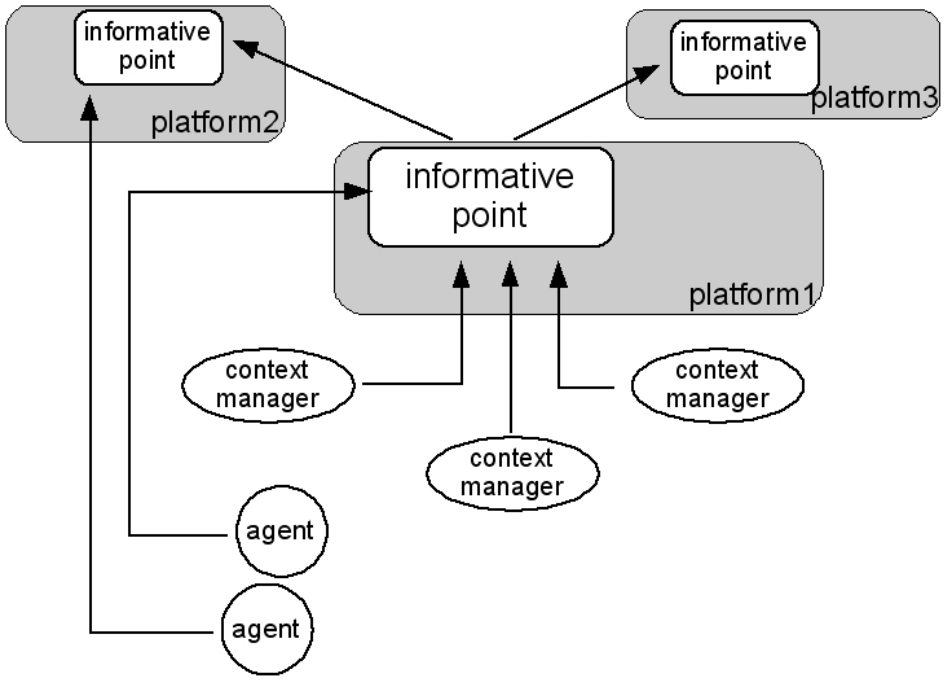

Figure 4.2: Links among informative points and other entities of the agent system

AGENT LIST	
Agent	Context Group

CONTEXT GROUP LIST	
Context Group	Manager

Table 4.1: Lists managed by an informative point

gets trustworthiness information from the context manager the agent belongs to (Table 4.1).

An informative point has a list of known points too. When a request is related to an unknown agent, the informative point provides the requester with the list of the other points it is linked to. Hence the requester can query the provided set of informative points.

In (Figure 4.3) it is summarized the query protocol for an information request coming from an agent that is member of a context group:

1. the agent queries the context manager of the group it belongs to in order to get reputation information about another agent;

2. if the query regards an unknown agent, the manager replies with the identity of the informative point to be queried, otherwise it provides requested agent information;

3. the agent can query the informative point
4. if the requested agent identity is in the list of known agents, the informative point ask for the related trustworthiness information to specific the context manager, otherwise if unknown it replies with a list of informative points it is linked to;
5. the agent can query the informative points in the received list.

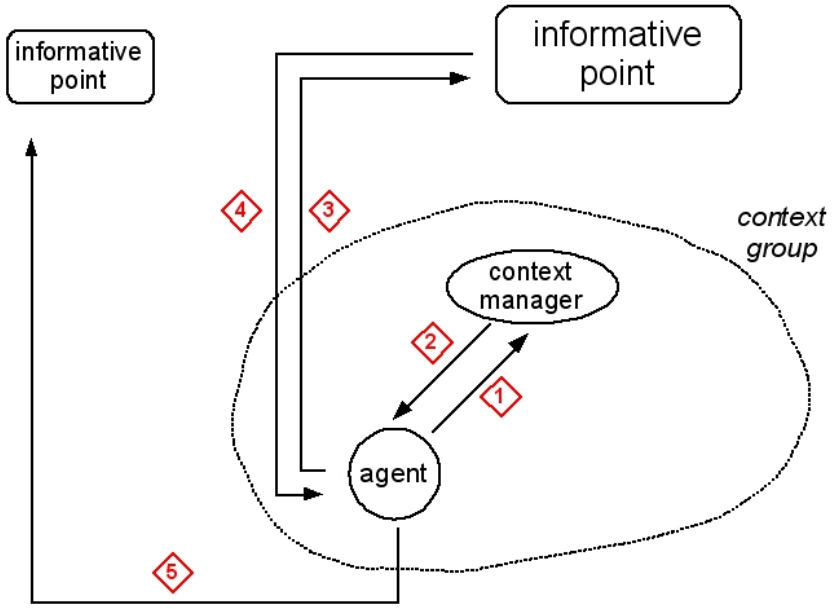

Figure 4.3: Query protocol

Hence an agent in a context group can get context reputation and social trustworthiness information from the manager. When an agent queries an informative point, it can get just social trustworthiness information regarding other agents. Social trustworthiness is a general evaluation, whereas context reputation is strictly related to a group.

The social structure that we devised with the informative points is depicted in Figure 4.4: the base layer represents the community of agents without any kind of social structure. Introducing the context groups, we lay a first level of social infrastructure on the system, i.e. the community participants are organized into context of interests. The second layer composed of informative points organizes the groups in order to collect information contained in the below layer.

Figure 4.4 points out the path followed by reputation and trustworthiness information: it is produced by agents with opinions, and through context groups and informative points it can get to other agents at the same bottom social level.

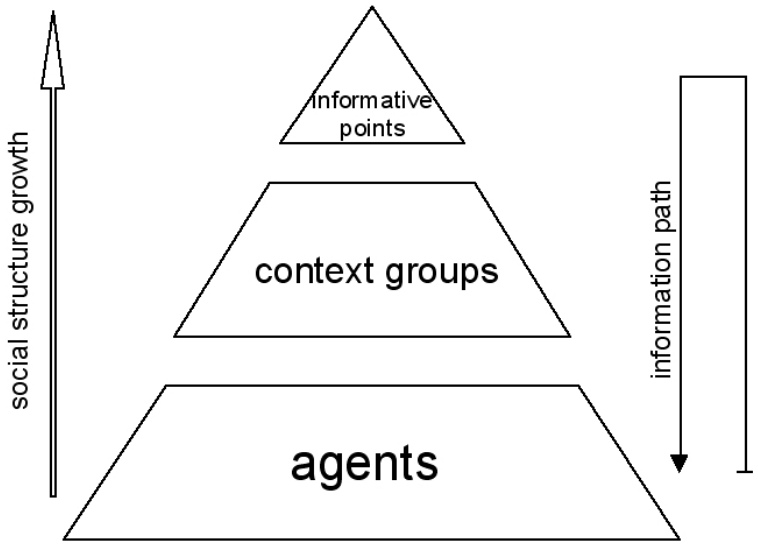

Figure 4.4: Schema of the social structure

Agents have been said to be autonomous entities; this key characteristic can be efficiently exploited when the system provides services that support the spread of information related to potential interaction partners and resources that are present in the society of agents. Autonomy remains an abstract concept if an agent is not supported by mechanisms that open its range of interaction out of the origin platform. Research on methodologies for agent-oriented software engineering focuses on the development of embedded infrastructure in order to model the environment where agents are supposed to interact [18].

Our social infrastructure reduces the lack of services that support sociality in the agent technology and reduces anonymity too.

Chapter 5

Conclusions

This thesis presented a model of a reputation management system that supports agents of multi-agent systems in trust-based decision making that are involved in their interactions.
The work in this thesis addresses issues that arise in open multi-agent system where agents could interact with unknown agents belonging to unknown agent systems. This implies security threats that could affect both agents and platforms.

In Chapter 1 we described the design architecture of an agent system and its key features. The agent programming paradigm is the evolution of the object-oriented programming paradigm. Agents are autonomous software processes which are aimed at achieving the goal of their owners. Agents can migrate from host to host to exploit available resources, and can communicate through messages with other agents to request services, information, etc. We described how heterogeneity issues among different implemented platforms are addressed with standard specifications that have been developed by FIPA as architectural guidelines for agent systems. We also introduced security threats that affect agent systems and actual implemented solutions.

In Chapter 2 we described emerging IT open systems' features and requirements. The agent paradigm suits open systems but agents require new security mechanisms because the actual mechanisms do not fit the features of open systems. Trust and reputation concepts that are employed in virtual marketplaces are a promising solution for security issues. The computational trust concept is derived from the trust concept used in sociology, psychology, philosophy, that is the same concept that people use in their everyday life. Trust is a belief about the willingness of another entity to honestly and fairly behave. It is based on previous interaction evidence or on recommendation from third parties.
For computational purposes, trust needs to be quantified. We described some trust models to show how trust is measured in actual systems. Reputation of an entity is the a value that expresses what is the degree of trust regarding this entity that is computed

from trust opinions of a community. A reputation system collects trust rates and computes reputation values for the evaluated entities.
A reputation system supports entities in trust-based decision-making when they interact with other entities in order to better evaluate risks coming from interactions. We described the features of actual implemented reputation systems.

In Chapter 3 we described in deep details the definition of ontology, and how they can be used as a common language in the communication among agents. We then explained the centralized architecture of the proposed reputation management model for soft security in multi-agent systems. The model is based on coalitions of agents in context groups. A context group is managed by a central manager that collects opinions in relation to its specific context regarding the behaviors of agents. A group is composed of agents that are interested in its specific context. Agents are not required to share trust information among them, because they just refer to the manager. This reduces message exchanging among agents.
The model defines two kinds of trustworthiness: the social trustworthiness is related to the general behavior of an agent, its general honesty and fairness; the context reputation is specifically related to the agent's behavior within the context, its dependability and reliability. To express opinions on the behaviors related to the context, the manager provides the members of the group with a context specific ontology that is aimed at enhancing the details of the opinions in order to improve its reliability and to reduce the generality. This in turn improves reliability of reputation values too.
The context reputation and the social trustworthiness values of an agent are calculated with a weighted average of the received opinions. The model is based on a simple algorithm to evaluate reputation from opinions, nonetheless it provides the agents with detailed information about the opinions that contributed to the computation.
Unfair ratings provided by malicious entities are not easy to detect. It is still an open issue in reputation systems. Our model allows agents to join a group only if they get a recommendation from a member, hence agents that are not trusted by anyone in the group cannot participate. The constrained access reduces potential malicious agents to bias reputation values. Ratings are also weighted according to the social trustworthiness of each agent, hence if we assume that malicious agents have low trustworthiness rates, their opinions lightly affect reputation values.

In Chapter 4 we analyzed the proposed reputation model from the social perspective. We described how a network of informative points in the community of agents can connect context groups in order to support agents searching for information that regards the honesty and reliability of other agents. The social structure composed of context groups and informative points allows agents to easily gather information that is useful when they decide to interact with others.

In conclusion, this thesis proposes a model for a reputation system to be employed in open multi-agent systems in order to provide agents with a soft security mechanism. Our context-based approach removes the generality of trust values that typically affects reputation systems. It was developed taking into account the general characteristics of the agent model. Its centralized approach suits the agent model since reputation communication involves each agent and the manager: agents are not required to gather information from other agents, hence even in highly dynamic agent systems, where they rapidly enter and leave hosts, the reputation system is not affected. The model fits also agent low communication bandwidth conditions because it does not require message exchanging among agents. The central management in each group prevents the agent to embed reputation computation functions and to manage other agents' opinions, hence the model fits low computational power agents. Agents can also join several context groups without affecting their computational load because a group can be considered as a repository for context opinions and a source of reputation information.

The reputation system model that was described in this thesis supports agents in trust-based decision making. It provides agents with information related to evaluations of both their honesty and their specific context group reputation. Agents can request information about agents involved in their interactions in order to reduce risks that could come from malicious behaviors. It can be effectively employed in every agent system because the model was designed in an agent-centric fashion in order to comply with the key features of those systems, and considering social requirements in order to efficiently share information among all the agents in the society regardless of its size.

Bibliography

[1] Jade: Java agent development framework. `jade.tilab.com`.

[2] *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*. ACM, 2002.

[3] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6*, page 6007, Washington, DC, USA, 2000. IEEE Computer Society.

[4] Amazon. `www.amazon.com`.

[5] Whitby Andrew, Jøsang Audun, and Indulska Jadwiga. Filtering out unfair ratings in bayesian reputation systems. In *The Third International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS2004), New York, USA*. ACM, July 2004.

[6] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. Jade: A software framework for developing multi-agent applications. lessons learned. *Inf. Softw. Technol.*, 50(1-2):10–21, 2008.

[7] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[8] Carole Bernon, Marie Pierre Gleizes, Sylvain Peyruqueou, and Gauthier Picard. Adelfe: A methodology for adaptive multi-agent systems engineering. In Petta et al. [67], pages 156–169.

[9] Cristian Bertocco and Carlo Ferrari. Context-dependent reputation management for soft security in multi agent systems. In *WI-IAT 2008: Proceedings of the International Joint Conference on Web Intelligence and Intelligent Agent Technology, December 9–12*, 2008.

[10] Walter Binder and Volker Roth. Secure mobile agent systems using java: where are we heading? In *SAC*, pages 115–119. ACM, 2002.

[11] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In Bruce Christianson, Bruno Crispo, William S. Harbison, and Michael Roe, editors, *Security Protocols Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer, 1998.

[12] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society, 1996.

[13] Niklas Borselius. Security in multi-agent systems. In Y. Mun and H. R. Arabnia, editors, *SAM 2002*, pages 31–36. CSREA Press, 2002.

[14] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[15] Ciarán Bryce, Paul Couderc, Jean-Marc Seigneur, and Vinny Cahill. Implementation of the secure trust engine. In Peter Herrmann, Valérie Issarny, and Simon Shiu, editors, *iTrust*, volume 3477 of *Lecture Notes in Computer Science*, pages 397–401. Springer, 2005.

[16] S. Buchegger and J. Le Boudec. A robust reputation system for mobile ad hoc networks, 2003.

[17] Sonja Buchegger, Cédric Tissières, and Jean-Yves Le Boudec. A test-bed for misbehavior detection in mobile ad-hoc networks – how much can watchdogs really do? In *WMCSA*, pages 102–111. IEEE Computer Society, 2004.

[18] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Engineering mobile agent applications via context-dependent coordination. *IEEE Trans. Software Eng.*, 28(11):1039–1055, 2002.

[19] V. Cahill, B. Shand, E. Gray, C. Bryce, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nicon, G. di Marzo Serugendo, J.-M. Seigneur, M. Carbone, K. Krukow, C. Jensen, Y. Chen, and M. Nielsen. Using trust for secure collaboration in uncertain environments, 2003.

[20] Sara Casare and Jaime Sichman. Towards a functional ontology of reputation. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 505–511, New York, NY, USA, 2005. ACM.

[21] Antonio Chella, Massimo Cossentino, Luca Sabatucci, and Valeria Seidita. Agile passi: An agile process for designing agents. *Comput. Syst. Sci. Eng.*, 21(2), 2006.

[22] Yang-Hua Chu, Joan Feigenbaum, Brian A. LaMacchia, Paul Resnick, and Martin Strauss. Referee: Trust management for web applications. *Computer Networks*, 29(8-13):953–964, 1997.

[23] Elizabeth Churchill, Les Nelson, and Diana K. Smetters. Useful computer security. *IEEE Internet Computing*, 11(3):10–12, 2008.

[24] D'Agents. `agent.cs.dartmouth.edu`.

[25] Chrysanthos Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *ACM Conference on Electronic Commerce*, pages 150–157, 2000.

[26] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[27] eBay. `www.ebay.com`.

[28] Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In Yves Demazeau, editor, *ICMAS*, pages 128–135. IEEE Computer Society, 1998.

[29] FIPA: Foundation for Intelligent Physical Agents. `www.fipa.org`.

[30] Foundation for Intelligent Physical Agents. *FIPA Ontology Service Specification*, 2001.

[31] Foundation for Intelligent Physical Agents. *FIPA Agent Management Specification*, 2004.

[32] Stan Franklin and Arthur C. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Jörg P. Müller, Michael Wooldridge, and Nicholas R. Jennings, editors, *ATAL*, volume 1193 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 1996.

[33] Diego Gambetta. Can we trust trust. In *Trust: Making and Breaking Cooperative Relations*, pages 213–237. Basil Blackwell, 1988.

[34] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4), 2000.

[35] Tyrone Grandison and Morris Sloman. Trust management tools for internet applications. In Paddy Nixon and Sotirios Terzis, editors, *iTrust*, volume 2692 of *Lecture Notes in Computer Science*, pages 91–107. Springer, 2003.

[36] Elizabeth Gray. *A Trust-Based Reputation Management System*. PhD thesis, Dept. of Computer Science, Trinity College Dublin, apr 2006.

[37] Robert S. Gray, David Kotz, George Cybenko, and Daniela Rus. D'agents: Security in a multiple-language, mobile-agent system. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 154–187. Springer, 1998.

[38] Mark Greaves, Victoria Stavridou-Coleman, and Robert Laddaga. Guest editors' introduction: Dependable agent systems. *IEEE Intelligent Systems*, 19(5):20–23, 2004.

[39] The OMG Object Management Group. `www.omg.org`.

[40] Thomas R. Gruber and Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[41] Tom Gruber. Ontology. In *Entry in the Encyclopedia of Database Systems, Ling Liu and M. Tamer Ozsu (Eds.)*. Springer-Verlag, 2008.

[42] Nicola Guarino. Understanding, building and using ontologies. *Int. J. Hum.-Comput. Stud.*, 46(2-3):293–310, 1997.

[43] Nicola Guarino. Formal ontology and information systems. In *International Conference on Formal Ontology in Information Systems (FOIS98)*, pages 3–15, Trento, Italy, June 1998. IOS Press.

[44] Nicola Guarino and Pierdaniele Giaretta. Ontologies and knowledge base: Towards a terminological clarification. *N.J.I. Mars (ed.) Towards Very Large Knowledge Bases*, pages 25–32, 1995.

[45] Vandana Gunupudi and Stephen R. Tate. Sagent: a security framework for jade. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 1116–1118. ACM, 2006.

[46] Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2004.

[47] Trung Dong Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006.

[48] Aglets IBM. `www.trl.ibm.com/aglets`.

[49] Audun Jøsang. An algebra for assessing trust in certification chains. In *NDSS*. The Internet Society, 1999.

[50] Audun Jøsang. A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–212, 2001.

[51] Audun Jøsang, Elizabeth Gray, and Michael Kinateder. Simplification and analysis of transitive trust networks. *Web Intelligence and Agent Systems*, 4(2):139–161, 2006.

[52] Audun Jøsang and Jochen Haller. Dirichlet reputation systems. In *ARES*, pages 112–119. IEEE Computer Society, 2007.

[53] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[54] Audun Jøsang and Roslan Ismail. The beta reputation system. In *In Proceedings of the 15th Bled Electronic Commerce Conference*, 2002.

[55] Thomas Juan, Adrian R. Pearce, and Leon Sterling. Roadmap: extending the gaia methodology for complex open systems. In *AAMAS* [2], pages 3–10.

[56] Michael Luck. Guest editorial: Challenges for agent-based computing. *Autonomous Agents and Multi-Agent Systems*, 9(3):199–201, 2004.

[57] Michael Luck, Peter McBurney, and Chris Preist. A manifesto for agent technology: Towards next generation computing. *Autonomous Agents and Multi-Agent Systems*, 9(3):203–252, 2004.

[58] Qusay H. Mahmoud and Leslie Yu. Making software agents user-friendly. *IEEE Computer*, 39(7):94–96, 2006.

[59] Antonio Maña, Antonio Muñoz, and Daniel Serrano. Towards secure agent computing for ubiquitous computing and ambient intelligence. In Jadwiga Indulska, Jianhua Ma, Laurence Tianruo Yang, Theo Ungerer, and Jiannong Cao, editors, *UIC*, volume 4611 of *Lecture Notes in Computer Science*, pages 1201–1212. Springer, 2007.

[60] Steve Marsh. *Formalizing trust as a Computational Concept*. PhD thesis, Dept. of Mathematics and Computer Science, University of Stirling, 1994.

[61] Peter McBurney and Michael Luck. The agents are all busy doing stuff! *IEEE Intelligent Systems*, 22(4):6–7, 2007.

[62] Peter McBurney and Simon Parsons. Engineering democracy in open agent systems. In Andrea Omicini, Paolo Petta, and Jeremy Pitt, editors, *ESAW*, volume 3071 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2003.

[63] Peter Mcburney and Simon Parsons. Engineering democracy in open agent systems. In *Engineering Societies in the Agents World, LNAI*. Springer, 2004.

[64] Lik Mui, Ari Halberstadt, and Mojdeh Mohtashemi. Evaluating reputation in multi-agents systems. In Rino Falcone, K. Suzanne Barber, Larry Korba, and Munindar P. Singh, editors, *Trust, Reputation, and Security*, volume 2631 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2002.

[65] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a metamodel for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[66] Kouichi Ono and Hideki Tai. A security scheme for aglets. *Softw., Pract. Exper.*, 32(6):497–514, 2002.

[67] Paolo Petta, Robert Tolksdorf, and Franco Zambonelli, editors. *Engineering Societies in the Agents World III, Third International Workshop, ESAW 2002, Madrid, Spain, September 16-17, 2002, Revised Papers*, volume 2577 of *Lecture Notes in Computer Science*. Springer, 2003.

[68] Lars Rasmusson, Andreas Rasmusson, and Sverker Janson. Reactive security and social control. In *In Proceedings, 19 th National Information Systems Security Conference*, 1996.

[69] Jordi Sabater. Evaluating the regret system. *Applied Artificial Intelligence*, 18(9-10):797–813, 2004.

[70] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 194–195, New York, NY, USA, 2001. ACM.

[71] Jordi Sabater and Carles Sierra. Reputation and social network analysis in multiagent systems. In *AAMAS* [2], pages 475–482.

[72] Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24(1):33–60, 2005.

[73] Glenn Shafer. Perspectives on the theory and practice of belief functions. *Int. J. Approx. Reasoning*, 4(5-6):323–362, 1990.

[74] Carles Sierra. Agent-mediated electronic commerce. *Autonomous Agents and Multi-Agent Systems*, 9(3):285–301, 2004.

[75] Carles Sierra, Jordi Sabater, Jaume Agustí-Cullell, and Pere Garcia. Integrating evolutionary computing and the sadde methodology. In *AAMAS*, pages 1116–1117. ACM, 2003.

[76] JACK Autonomous Software. `www.aosgrp.com`.

[77] Jadex Agent System. `vsis-www.informatik.uni-hamburg.de/projects/jadex`.

[78] Tacoma. `www.tacoma.cs.uit.no`.

[79] Himanshu Tagra and Saroj Kaushik. Mole agents - a new phenomenon in auctions. In *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06)*, pages 439–443, Washington, DC, USA, 2005. IEEE Computer Society.

[80] Robert A. Wilson and Frank C. Keil, editors. *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. Bradford Books. MIT Press, September 2001.

[81] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[82] Li Xiong and Ling Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *CEC*, pages 275–284. IEEE Computer Society, 2003.

[83] Li Xiong and Ling Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *ACM Conference on Electronic Commerce*, pages 228–229. ACM, 2003.

[84] Bin Yu and Munindar P. Singh. An evidential model of distributed reputation management. In *AAMAS* [2], pages 294–301.

[85] Weiwei Yuan, Donghai Guan, Sungyoung Lee, Young-Koo Lee, and Heejo Lee. Filtering out unfair recommendations for trust model in ubiquitous environments. In Aditya Bagchi and Vijayalakshmi Atluri, editors, *ICISS*, volume 4332 of *Lecture Notes in Computer Science*, pages 357–360. Springer, 2006.

[86] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.

[87] Franco Zambonelli and Andrea Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, 2004.

[88] Franco Zambonelli and H. Van Dyke Parunak. Signs of a revolution in computer science and software engineering. In Petta et al. [67], pages 13–28.