Social Knowledge in Multi-Agent Systems

by

Pavel Tichý

Submitted to the Faculty of Electrical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Artificial Intelligence and Biocybernetics
at the Department of Cybernetics of
the Czech Technical University in Prague

December, 2003
Social Knowledge in Multi-Agent Systems

Disertační práce

Ing. Pavel Tichý

Praha, 2003

katedra: Kybernetiky
obor: Umělá inteligence a biokybernetika
Contents

1 INTRODUCTION ........................................................................................................ 1
  1.1 DEFINITION OF TERMS ...................................................................................... 1
    1.1.1 Agent .............................................................................................................. 1
    1.1.2 Multi-Agent System .......................................................................................... 2
  1.2 THESIS GOALS .................................................................................................. 3
  1.3 THESIS ORGANIZATION ..................................................................................... 3

2 SOCIAL KNOWLEDGE DESCRIPTION .................................................................. 5
  2.1 SOCIAL KNOWLEDGE TYPES .......................................................................... 5
  2.2 FORMAL DESCRIPTION OF INTERACTIONS IN MAS ...................................... 8
    2.2.1 Agent UML ...................................................................................................... 8
    2.2.1.1 UML Sequence Diagrams ............................................................................ 8
    2.2.1.2 UML Collaboration Diagrams ..................................................................... 9
    2.2.2 External Agent-Object-Relationship Models .................................................. 9

3 MULTI-AGENT SYSTEM ARCHITECTURES ......................................................... 11
  3.1 STATIC ARCHITECTURES .................................................................................. 12
    3.1.1 Hierarchical Architecture .............................................................................. 12
    3.1.2 Autonomous Architecture ............................................................................. 13
  3.2 DYNAMIC ARCHITECTURES ............................................................................ 13
    3.2.1 Broadcasting .................................................................................................. 14
    3.2.2 Federated Architectures .................................................................................. 15
      3.2.2.1 Matchmaker .............................................................................................. 16
      3.2.2.2 Broker ...................................................................................................... 17
      3.2.2.3 Mediator .................................................................................................... 19
      3.2.2.4 Blackboard .............................................................................................. 20
      3.2.2.5 Monitor ...................................................................................................... 21
      3.2.2.6 Facilitator .................................................................................................. 22
      3.2.2.7 Embassy .................................................................................................... 22
      3.2.2.8 Anonymizer .............................................................................................. 23
      3.2.2.9 Job Agency ............................................................................................... 23
    3.2.3 Acquaintance Models .................................................................................... 24
    3.2.4 Architecture without Middle-Agents ............................................................. 25
    3.2.5 Categorization of Dynamic Architectures ..................................................... 28

4 SPECIFIC MANAGEMENT TECHNIQUES FOR SOCIAL KNOWLEDGE.......... 31
  4.1 SOCIAL KNOWLEDGE IN MAS STANDARDS ................................................. 31
    4.1.1 Knowledge Query and Manipulation Language (KQML) ................................. 32
    4.1.2 Foundation for Intelligent Physical Agents (FIPA) ........................................ 33
    4.1.3 Language for Advertisement and Request for Knowledge Sharing (LARKS) .... 34
    4.1.4 Service Description Language (SDL) ............................................................. 35
    4.1.5 PHOSPHORUS ............................................................................................... 35
    4.1.6 DAML-S .......................................................................................................... 36
    4.1.7 Service Location Protocol (SLP) ..................................................................... 36
  4.2 SOCIAL KNOWLEDGE MANAGEMENT IN MAS IMPLEMENTATIONS .......... 37
    4.2.1 Java Agent DEvelopment Framework (JADE) ................................................. 38
    4.2.2 FIPA Open Source (FIPA-OS) ....................................................................... 38
    4.2.3 Production Planning Tool (ProPlanT) .............................................................. 39
    4.2.4 SHAred Dependency Engineering (SHADE) and COmmon INterest Seeker (COINS) ... 39
    4.2.5 InfoSleuth ....................................................................................................... 40

5 SOCIAL KNOWLEDGE DISTRIBUTION ......................................................... 41
  5.1 CENTRALIZED SOCIAL KNOWLEDGE ............................................................ 41
    5.1.1 Blackboard Architecture ................................................................................. 42
NEW CLASSIFICATION OF FAILURES IN MAS AND FAILURE IMPACT ........... 50
7.1  AGENT FAILURE TYPES........................................................................... 50
7.2  MULTI-AGENT SYSTEM FAILURE TYPES...................................................... 54
7.3  STATIC IMPACT OF MIDDLE-AGENT FAILURE.............................................. 56
7.4  DYNAMIC IMPACT OF MIDDLE-AGENT FAILURE........................................ 59

NEW STRUCTURE OF SOCIAL KNOWLEDGE DISTRIBUTION ............... 61
8.1  DYNAMIC HIERARCHICAL TEAMS................................................................. 62
8.1.1  Dynamic Hierarchical Teams Architecture Description ......................... 62
8.1.2  Fault Tolerance in Dynamic Hierarchical Teams....................................... 64
8.1.3  Social Knowledge Management in Dynamic Hierarchical Teams .............. 66
8.1.3.1  Breadth Knowledge Propagation .............................................................. 67
8.1.3.2  Depth Knowledge Propagation ................................................................. 70
8.1.3.3  No Knowledge Propagation ..................................................................... 72
8.1.3.4  Knowledge Propagation on Demand....................................................... 75
8.1.3.5  Knowledge Caching.................................................................................... 77
8.1.4  Scalability of Dynamic Hierarchical Teams................................................ 79
8.1.5  Reconfiguration in Dynamic Hierarchical Teams....................................... 81
8.1.6  Failure Detection Mechanisms in Dynamic Hierarchical Teams................ 82
8.1.6.1  Response Timeout Mechanism................................................................. 82
8.1.6.2  Heartbeat Mechanism .............................................................................. 83
8.1.6.3  Meta-Agent Observation......................................................................... 84

EXPERIMENTAL PART................................................................................. 85
9.1  COMPARISON OF KNOWLEDGE PROPAGATION METHODS .................... 85
9.1.1  Comparison by the Number of Messages.................................................. 86
9.1.2  Comparison by Total Running Time......................................................... 92
9.1.3  Comparison by Communication Frequency............................................. 95
9.2  EXPERIMENTS WITH ROBUSTNESS........................................................... 97
9.2.1  Robustness in the Dynamic Hierarchical Teams....................................... 97
9.2.1.1  Hierarchy without Redundancy................................................................. 98
9.2.1.2  Redundant Hierarchy.............................................................................. 99
9.3  COMPARISON OF SOCIAL KNOWLEDGE DISTRIBUTIONS................... 101
9.4  SUMMARY OF RESULTS............................................................................... 102

CONCLUSIONS.............................................................................................. 104
10.1  FULFILLMENT OF GOALS.......................................................................... 104
10.2  SUMMARY OF RESULTS............................................................................ 104
10.3  CONTRIBUTION.......................................................................................... 106

FUTURE WORK............................................................................................. 107
List of Figures

Figure 1: AUML sequence diagram example ............................................................. 8
Figure 2: Concurrency in UML sequence diagrams ................................................ 9
Figure 3: AUML collaboration diagram example .................................................... 9
Figure 4: Examples of interaction types in the External AOR model ...................... 10
Figure 5: Simplified External AOR interaction sequence diagram example .......... 10
Figure 6: Hierarchical three-layer architecture example ..................................... 12
Figure 7: Autonomous architecture example ...................................................... 13
Figure 8: Service matchmaking .......................................................................... 16
Figure 9: Service brokering ................................................................................. 18
Figure 10: Service recruiting .............................................................................. 19
Figure 11: Synchronous blackboard architecture example .................................. 21
Figure 12: Asynchronous blackboard architecture example ................................ 21
Figure 13: Job agency mechanism ...................................................................... 23
Figure 14: Connectivity graph of a planar rectangular lattice structure with a shortcut ............................................................... 26
Figure 15: Service matchmaking in the KQML example .................................. 32
Figure 16: Service brokering in the KQML example ......................................... 33
Figure 17: Categorization of agent failures ......................................................... 51
Figure 18: Example of 2-level DHT architecture ................................................. 63
Figure 19: Example of 3-level DHT architecture and associated structure of teams ............................................................... 63
Figure 20: Breadth knowledge propagation example .......................................... 69
Figure 21: Depth knowledge propagation example .............................................. 71
Figure 22: Knowledge propagation on demand applied to the depth knowledge propagation ............................................................... 75
Figure 23: Knowledge propagation on demand applied to the no knowledge propagation ............................................................... 77
Figure 24: Knowledge caching example ................................................................ 78
Figure 25: Dynamic reconfiguration of the structure of middle-agents as reaction to a failure of a middle-agent ................................................................. 81
Figure 26: Test environment for comparison of knowledge propagation methods ........................................................................................................ 85
Figure 27: Test case 1 of knowledge propagation methods for the number of messages .................................................................................................. 87
Figure 28: Message flow example for the search in the no knowledge propagation method .......................................................................................... 88
Figure 29: Example of search using the no knowledge propagation method .......... 89
Figure 30: Message flow example for the search in the depth knowledge propagation method .......................................................................................... 89
Figure 31: Theoretical comparison of knowledge propagation methods ............... 91
Figure 32: Probability distributions and probabilities used to compute $P_F$ ............ 91
Figure 33: Test case 1 of knowledge propagation methods for the total running time ........................................................................................................ 92
Figure 34: Percent errors of knowledge propagation methods in test case 1 ............ 94
Figure 35: Communication frequency of knowledge propagation methods for $P_S = 0.5$ .................................................................................................. 96
Figure 36: The percent errors of knowledge propagation methods ....................... 97
Figure 37: One no response failure of the global middle-agent in the hierarchical architecture .......................................................................................... 98
Figure 38: Testing architecture for a robustness testing ....................................... 99
Figure 39: Two no response failures of the global middle-agents in the DHT architecture with three members of the team ............................................. 100
List of Tables

Table 1: Dynamic architectures categorized by initial privacy concerns ............... 29
Table 2: Fixed scalability of various types of social knowledge distributions ........ 80
Table 3: Failure impact and redundancy of various types of social knowledge distributions ................................................................. 101
Acknowledgements

First and foremost, I would like to thank my wife Petra, who encouraged me and believed in me during the long path to my PhD. Thank you for your patience.

I would like to thank my family who helped shape the person I am now. Thank you for your support in all possible ways.

I would like to thank Prof. Vladimír Mařík who guided and supported me during nearly all of my university and professional part of life. Your critical eye, ability to immediately see the root of a problem, and ability to vigorously protect and encourage the people you lead are just a few of your many excellent qualities.

I want to thank Associate Prof. Jiří Lažanský, my advisor, who supported me all the way during my PhD, ready to help any time.

I want to thank Prof. Marie Demlová for her wise suggestions and very detailed proofreading of the graph theory section, and also Associate Prof. Eduard Krajník and Dr. Petr Hellinger for their valuable suggestions in mathematics.

I would like to thank everyone at the Czech Technical University in Prague who made this work possible, especially Dr. Michal Pechouček who showed me a light at the end of the tunnel several times.

I would like to thank my colleagues from Rockwell Automation, Research Center Prague, especially Filip Macůrek for his detailed proofreading of the draft of my PhD and Petr Šlechta for sharing his clever ideas and opinions.

I want to thank my colleagues from Rockwell Automation Cleveland and Rockwell Scientific Company for allowing me to work with them in the field of multi-agent systems for industrial applications while supporting me throughout my research.

I would like to thank Dr. Raymond Staron who carefully proofread the whole 40,000 word thesis line by line and corrected my English.
1 Introduction

The research and development of multi-agent systems (MAS) became very popular in the last decade. The distribution, robustness, and high-level knowledge utilization of multi-agent systems make them quite suitable for distributed, complex, flexible, and open systems. Multi-agent systems are applicable in a wide range of applications starting from distributed industrial systems where multi-agent systems offer a high-level control over a low-level execution, through the multi-agent systems used for planning and scheduling, to very open systems running on the Internet.

All these systems have something in common - social knowledge. Briefly, social knowledge is knowledge about agents in the multi-agent system. With this knowledge, an agent can locate any other agent in the system, communicate with it, and enter into collaborative relationships with it. In this thesis we focus mainly on distribution, management, and robustness of social knowledge.

1.1 Definition of Terms

Terms such as agent and multi-agent system have been used in the introductory paragraphs without precise definitions. We will focus on these definitions in following sections.

Other terms are defined when they are first used in this document. We also assume that the reader has at least basic knowledge in the area of computer systems, mathematics, and multi-agent systems. Definitions for the remaining terminology can be found for instance in [144] and [19].

1.1.1 Agent

Many researchers use the term agent very frequently, but there is no definition on which even a majority of researchers agree. There is a lack of universally accepted definitions [47]. Also most agent developers have their own opinion on exactly what constitutes an agent and no two developers appear to share exactly the same opinion [145]. Or, that the definition of an agent directly depends on the application [68]. Sometimes even the same author uses different definitions as the definition evolves over time and with gained experience.

Moreover, there are many variations of the term agent, e.g., software agent [30], intelligent agent [128], and autonomous agent [47].

- ‘An agent is a self-contained problem-solving system capable of autonomous, reactive, proactive, and social behavior’ - Wooldridge and Jennings [145].
‘An agent is a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives’ - Jennings, Sycara, and Wooldridge [47].

‘An agent is an object that can say go (dynamic autonomy) and no (deterministic autonomy)’ - Odell, Parunak, and Bauer [88].

‘An agent is described as an entity that perceives and acts’ - Wickler and Tate [138].

‘An intelligent software agent is a program that acts on behalf of their human users in order to perform laborious information gathering tasks, such as locating and accessing information from various on-line information sources, resolving inconsistencies in the retrieved information, filtering away irrelevant or unwanted information, integrating information from heterogeneous information sources and adapting over time to their human users’ information needs and the shape of the Infosphere’ - Sycara, Decker, and Pannu [128].

‘An agent is a software entity that has enough autonomy and intelligence to carry out various tasks with little or no human intervention’ - Wong and Sycara [140].

The definition of agent used in this thesis is:

‘An agent is an autonomous unit that is able to interact with its environment in an intelligent manner’ [75].

Critics of agents usually compare an agent with an object and argue that an agent and an object are the same entities. The main difference is in the degree to which agents and objects are autonomous. ‘An object does not exhibit control over its behavior’ is expressed in [47] followed by the slogan: ‘objects do it for free; agents do it for money’ and from the programming perspective ‘agents are each considered to have their own thread of control’.

1.1.2 Multi-Agent System

Jennings, Sycara, and Wooldridge [47] define an agent-based system first, as ‘the one in which the key abstraction used is that of an agent’, and then they define multi-agent system as a case of agent-based system that contains more than one agent.

The multi-agent system has to contain all of the following characteristics to distinguish it from other distributed systems [47]:

- each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
- there is no global system control;
- data is decentralized; and
• computation is asynchronous.

As a rule of thumb, a multi-agent system is often regarded as *large* if it contains more than 10 agents [145].

1.2 Thesis Goals

The goals of this thesis are:

• To design and implement a structure of social knowledge distribution in a multi-agent system that increases robustness of the whole system relative to existing structures while preserving scalability. The structure has to offer demanded level of fault tolerance.

• To design and implement various knowledge management techniques that can be applied on this structure of social knowledge distribution to cover different demands of agents for registration, unregistration, and social knowledge search.

• To use and compare these knowledge management techniques in practical experiments in a multi-agent system to verify applicability of these methods on a real system.

1.3 Thesis Organization

In this thesis we describe, in section 2, which types of information social knowledge covers. Then we briefly describe several social interaction modeling techniques that can be used in the area of design and presentation of social knowledge.

In section 3 we present an overview of different types of multi-agent system architectures and rigorously summarize them from the point of view of social knowledge.

An overview of standards in the area of multi-agent systems that significantly affect social knowledge management is presented in section 4, followed by a brief description of several frameworks for development of multi-agent systems. We also examine several types of multi-agent systems that use different social knowledge distribution and management techniques.

In section 5 we identify different types of social knowledge distributions. Centralized, distributed, and hybrid architectures are described from the point of view of social knowledge distribution.

In section 6 we focus on social knowledge robustness. We present known techniques to measure and increase robustness of agents and of the whole multi-agent system.

We propose a classification of agent failures, in section 7, from the point of view of an observer of the agent. We also describe different failure types of
the whole multi-agent system. We propose a set of middle-agent failure impact attributes, average redundancy, and weighted average redundancy measures that can be used to assess the static impact of a middle-agent failure on the whole multi-agent system. We describe how graph theory is used to deal with fault tolerance and we use it to evaluate the dynamic impact of middle-agent failure.

In section 8 we define a new structure of social knowledge distribution. We describe various social knowledge management techniques that we developed to increase the robustness relative to the hierarchical techniques and to increase the scalability relative to the distributed techniques.

Finally, in section 9, we examine proposed social knowledge distribution and management techniques in practical experiments to provide a comparison of these approaches based on different properties of the environment. We test the robustness of the new structure of social knowledge distribution and we compare various types of social knowledge distributions for robustness.
2 Social Knowledge Description

Prior to the description of social knowledge, it is necessary to specify the term knowledge. One of the approaches to describe knowledge is to compare it with data [139].

- ‘Data describes specific instances and events. Data may be gathered automatically or clerically. The correctness of data can be verified vis-à-vis the real world.’
- ‘Knowledge describes abstract classes. Each class typically covers many instances. Experts are needed to gather and formalize knowledge. Data can be used to disprove knowledge.’

Social knowledge in the multi-agent systems can be understood as knowledge that is used to deal with other agents in the multi-agent system. Social knowledge consists of the information about the name of agents [20], their location (address), their capabilities (services), the language they use, their actual state, their conversations, behavioral patterns, and so on [65]. Moreover, the information about beliefs, desires, and intentions of other agents in the system can also be part of social knowledge ([144], [92], and [93]).

Knowledge that is located in agents can be formally split into two distinct types [7]:

- **Domain knowledge** (or problem-solving knowledge [65]) - concerns a problem-solving domain and an environment of an agent. Domain knowledge represents the decision-making process of an agent.
- **Social knowledge** - allows an agent to interact with other agents and possibly improves this process.

In the following sections, we present and describe social knowledge types from different points of view. Then we present several social interaction-modeling techniques that can be used to design and present social knowledge.

2.1 Social Knowledge Types

In this section we present and describe social knowledge types from different points of view. All of the presented classifications of social knowledge are applicable also to general knowledge.

For social knowledge management it is advantageous to classify different types of social knowledge by the frequency of its change. Permanent information, for example, does not have to be updated. Only in special cases, however, is information about an agent permanent. An agent’s location can change in the case of...
mobile agents, and an agent’s name can also change, when for instance two agents merge into one.

The classifications of social knowledge are as follows.

- **Permanent** social knowledge is the information about other agent(s) in the system that never changes at run-time. The permanent knowledge usually consists of knowledge about a communication language, communication protocol, name, location of static agents, and so on.

- We define **semi-permanent** social knowledge as the information that changes rarely. Usually, semi-permanent social knowledge is information about the name, location, capabilities, and languages of an agent. Note that this may vary based on multi-agent system type. For instance, in the case of mobile agents, information about the location can change often.

- Finally we define **temporary** social knowledge as the information that changes often. Usually, the temporary social knowledge is the information about the state of other agents and about beliefs, desires, and intentions of other agents in the system [92] and [93].

Permanent and semi-permanent social knowledge is usually in a qualitative form. The temporary social knowledge is more in a quantitative form, i.e., it contains data that are usually described in a numerical form.

Social knowledge can be further classified by its format as

- **declarative** knowledge - a format that directly allows all kinds of knowledge management and reasoning techniques. The declarative knowledge is stored explicitly and does not imply its usage, which can result in unforeseen kinds of reasoning; and

- **procedural** knowledge - a format in which the knowledge is implicitly stored as an algorithm. Knowledge can be obtained by application of the algorithm.

An example of declarative social knowledge is ‘the agent named Conveyor7 has a capability to move items’. On the other hand, an example of procedural social knowledge is ‘if the name of an agent contains letters ‘Conveyor’ then the agent has a capability to move items’.

Social knowledge can be classified from a privacy aspect [104] as

- **public knowledge** - can be shared with any agent from the multi-agent system.

- **semi-private knowledge** - can be shared with an agent from the multi-agent system that is within the social neighborhood, i.e., a set of agents, which an agent keeps specific information about (see [104] for detailed description).

- **private knowledge** - is not shared with any other agent.
Moreover, the privacy aspect of social knowledge can be defined from the point of view of the whole multi-agent system.

- **Public knowledge** - can be shared with any multi-agent system.

- **Semi-private knowledge** - can be shared with a multi-agent system that is within the social neighborhood, i.e., a set of multi-agent systems, which a multi-agent system keeps specific information about.

- **Private knowledge** - is not shared outside of the multi-agent system.

Social knowledge, or any knowledge in general, can be classified from the point of view of its source [105] as:

- **Empirical knowledge** - the source is direct observation of the domain. An example of empirical social knowledge is the locations or capabilities of agents currently present in a multi-agent system.

- **Theoretical knowledge** - based on scientific laws and principles. An example of theoretical social knowledge is the knowledge about a protocol that is used for communication among agents. This knowledge is usually not gathered dynamically from the observation of the communication among agents, but is known a priori.

Another classification of knowledge, by its orientation, can be used to classify social knowledge [105]:

- **Object-level knowledge** (knowing what) - the knowledge about properties and behavior of objects. The social knowledge example is again the knowledge about other agents.

- **Heuristic knowledge** (knowing what if) - the knowledge about mutual relationships among instances of the object-level knowledge. The social knowledge example is the knowledge that provides the information about the physical location of an agent if the address of the agent is known.

- **Meta-level knowledge** (knowing how) - the knowledge that is used to reason about other knowledge. The social knowledge example is the knowledge about how to choose the most reliable agents with a particular capability, assuming that the reliability is not an attribute of an agent that is known a priori.

In the previous sections we showed where social knowledge could reside and how types of social knowledge can be distinguished. Next, we show how abstraction functions can help to improve social knowledge representation.

---

1 There are multi-agent systems that use a special protocol offering agents that can be contacted and used to negotiate on behalf of agents that do not have the knowledge about a particular protocol. Nevertheless, these special agents are a priori preprogrammed to have the knowledge about these protocols.
2.2 Formal Description of Interactions in MAS

Modeling techniques are used not only for development of multi-agent systems, but also for the visualization and presentation of interactions among agents including interactions related to the social knowledge management. In the following sections we briefly describe several approaches that we are using to visually depict interactions among agents.

2.2.1 Agent UML

The Unified Modeling Language (UML) for object-oriented software development [88] has been gaining acceptance. The agent UML (AUML) addresses a growing concern for agent-based software. The AUML is currently in the stage of development as the work plan of FIPA (see section 4.1.2) and the final draft of an AUML specification is planned for March 2004.

The AUML creates additional elements to the UML diagrams to support the modeling of multi-agent systems and interactions among agents. The AUML in some diagrams adopts the UML completely; the additions are in the area of sequence diagrams, collaboration diagrams, activity diagrams, and statecharts. We will briefly describe the sequence diagrams and the collaboration diagrams because these diagrams are widely used for the visualization of communication among agents. Further details can be found in [88].

2.2.1.1 AUML Sequence Diagrams

For social interactions in which the sequence of interactions is an important aspect, the UML sequence diagrams are beneficial to use. The sequence of interactions is depicted as a top-down ordering of interactions, where the top is the beginning of the sequence and time flows from the top to the bottom.

![AUML sequence diagram example](image)

**Figure 1: AUML sequence diagram example**

The agents are listed within the rectangles at the top of the diagram, with descending vertical lines (see Figure 1). Agent identifications are depicted in a form agent-name/role:class, for instance Bob/Employee:Person, but the agent-name is not mandatory. An interaction is depicted as a horizontal arrow starting from the originating agent and leading to the destination agent. The arrow is supplemented by the description of the interaction.
There are three possible types of concurrent threads of an interaction in the AUML sequence diagrams:

a) *exclusive or* - exactly one communication act will be sent;

b) *inclusive or* - any possible number of communication acts will be sent concurrently (zero or more); and

c) *and* - all communication acts will be sent.

### 2.2.1.2 AUML Collaboration Diagrams

The AUML collaboration diagrams are used to depict patterns within an interaction among agents. The sequence of communication actions is provided only by the number associated with each action.

![AUML collaboration diagram example](image)

A connecting line depicts an interaction between two agents. The communication action is again depicted as an arrow associated with the line, annotated with the sequence number and a name for the action (see Figure 3). A dotted arrow with <<role change>> describes a change of role for an agent. The role change technique can be used also in the AUML sequential diagrams.

### 2.2.2 External Agent-Object-Relationship Models

The External Agent-Object-Relationship (AOR) model, described in more detail in [137], is used for more complex or dynamic modeling. The AOR model is an extension of the Unified Modeling Language. Interactions are sub-classified into three types and they can be depicted only as one directional (see Figure 4).
Social Knowledge in Multi-Agent Systems - Social Knowledge Description

1. Communicative
   - Inform the Price

2. Non-communicative
   - Send the Book

3. Non-action
   - Shelf Empty

**Figure 4: Examples of interaction types in the External AOR model**

1. *Communicative action* event (a dot-dashed line boundary) - the interaction is performed via a communication.

2. *Non-communicative action* event (a full line and full arrow boundary) - the interaction is performed by means other than a communication.

3. *Non-action* event (a full line and partial arrow boundary) - an event that does not originate from an action.

We are using two types of diagrams based on the following interaction types:

- *Interaction frame diagrams* - each interaction is drawn from the originating agent to the destination agent.

- *Interaction sequence diagrams* - depicts a social interaction process as a sequence of partial interactions. Each partial interaction is supplemented with a number within a circle to indicate its ordering within the entire interaction process.

If it is not necessary for model understanding, we will omit the distinction among interaction types and depict the interaction as an arrow supplemented by the description of the interaction (see Figure 5).

**Figure 5: Simplified External AOR interaction sequence diagram example**

All of the social interaction modeling techniques -- the AUML sequential diagrams, AUML collaboration diagrams, external AOR frame diagrams, and external AOR sequence diagrams -- can be used to depict the same interaction among agents, but in some cases one of them provides a better description over the others. Therefore, in the following chapters, we will choose the most advantageous modeling techniques for the description of interactions among agents.
3 Multi-Agent System Architectures

The multi-agent system architectures as software entities draw their inspiration from real world social structures. There are different types of society structures in the real world such as hierarchical, matchmaker, and broker structures. The multi-agent system architecture functionality is more simplified than the real world society structure, because in the real world there are people on different positions and they are able to learn, reason, adapt, optimize, etc. much more ‘intelligently’ than their counterparts in the multi-agent systems developed so far.

The hierarchical structure (see section 3.1.1) is mainly used in companies with a statically defined hierarchy of directors, managers, and workers. The hierarchy is modified as people move through their positions in the company, but the movement is very slow in comparison with task information passing through the hierarchy.

The matchmaker and the broker architectures (see sections 3.2.2.1 and 3.2.2.2) are basic types of dynamic architectures. On one hand, people are looking for services, and, on the other hand, people or organizations are offering services. The matchmakers or the brokers are in the middle and they offer both service registration and service lookup.

The real life social structures are usually more complicated than the multi-agent system architectures. They offer for instance learning mechanisms, which can significantly increase the efficiency of the social knowledge management process. This idea is followed by the mediator approach (see section 3.2.2.3) that is based on the matchmaker and the broker architectures with additional coordination roles.

The acquaintance model (see section 3.2.3) is trying to distribute social knowledge into all agents in the system. The acquaintance models are based on the idea that, in real life, service requestors are able to learn from previous contracts and keep knowledge about negotiation partners locally and up to date.

There is no one good organizational structure for all types of human organizations [17] and the same holds for multi-agent systems. Moreover, the increased scalability is obtained if the architecture of a multi-agent system can adapt its structure for various population sizes [131]. Therefore, we focus on a description of many different types of multi-agent system architectures to point out their advantages and disadvantages.

Common to all architectures is the existence of providers and requesters. The providers offer their services (usually described as capabilities [11]) and the requesters ask for the services.

First, we describe static architectures (see section 3.1), where the interconnections among agents are statically created at design time. Then we
focus on dynamic architectures (see section 3.2), where the organization is created dynamically at run-time.

3.1 Static Architectures

The idea to put social knowledge into the system at design time is common to all static architectures, where the structure of the system is a priori determined. Instead of the term static, the negation of the term openness can be used, where the term openness can be described by its fundamental aspect that an agent should not have a priori knowledge of other agents in the system [85].

We distinguish two types of static architectures so far: hierarchical and autonomous [115]. They are not open to new agents coming into or existing agents leaving the system, which is the main disadvantage of static architectures. Because a capability (service) of an agent never changes at run-time, any capability related communication is pruned away, which is a significant advantage of static architectures.

3.1.1 Hierarchical Architecture

In the hierarchical architecture of a multi-agent system each agent has a predefined role in the community. At design time, the system integrator creates a hierarchy of agents that at run-time becomes static. The hierarchical approach is mainly used in manufacturing, where it follows the traditional hierarchical organization of a manufacturing enterprise [115]. Typically a three-layer hierarchical architecture is used (see Figure 6), where boxes represent agents and arrows represent possible communication paths.

![Hierarchical three-layer architecture example](image)

The hierarchical static architecture is very advantageous in the case where there are hierarchical relations among agents, i.e., a parent agent uses only capabilities of own children. A disadvantage is the absence of fault-tolerance. In the case that a manager agent fails the whole section that belongs to this manager becomes unusable.
3.1.2 Autonomous Architecture

The autonomous architecture [115] or point-to-point architecture [61] is another type of static architecture. The autonomous architecture is used primarily when there are no obvious managers in the system and agents are able to communicate directly among themselves (see Figure 7). Similarly, as in the hierarchical architecture, the information about agent types, capabilities, and locations has to be known at design time and stored within each agent. Each agent has to keep this knowledge at run-time.

![Figure 7: Autonomous architecture example](image)

The advantage of the autonomous architecture is that there is not a single point of failure in this type of architecture. The disadvantage is that each agent has to know the social information about the whole system, which could be very large and time consuming to obtain. As a consequence the autonomous architecture is not scalable.

3.2 Dynamic Architectures

Social knowledge in the dynamic architectures is usually discovered during run-time by agents based on their capabilities and may change during its life cycle.

The key process of dynamic architectures is *matchmaking*, which can be defined many possible ways:

- ‘process of recommending agents that provide services to agents requesting services’ - Nodine, Bohrer, and Ngu [84].
- ‘process of finding an appropriate provider for a requester through a middle-agent’ - Sycara et al. [127], [123], [124], and [147] (middle-agent is defined in section 3.2.2).
- ‘protocol to find matched pairs \(< a_i, b_j >\) between two registered groups of participants \(A = (a_1, \ldots, a_m)\) and \(B = (b_1, \ldots, b_n)\), which satisfy \(< a_i, b_j > \in R\) for a special relation \(R\) we try to find’ - Lee and Kim [60].
- ‘facilitation services for content-directed routing and intelligent matching of information consumers and producers’ - McGuire et al. [78].
‘process of matchmaking allows an agent with some tasks, the requester, to learn the contact information and capabilities of another agent, the server, who may be able to execute some of the requester’s tasks’ - Sheory, Sycara, and Jha [117].

‘service that pairs agents seeking a particular service with agents that can perform that service’ - Bayardo et al. [4].

Beside the term matchmaking, other terms with very similar meaning are also used:

- *brokering*, for instance in [84],
- *connection problem*, for instance in [14],
- *service discovery problem* that is closely related to resource discovery problem, for instance in [32], and
- *mediating*, for instance in [98].

All of the mentioned terms were intended to describe the same general functionality, but the naming coincides with some specific functionality. For instance, the term matchmaking coincides with the functionality of the matchmaker - the service matchmaking (see section 3.2.2.1), the term brokering coincides with the functionality of the broker - the service brokering (see section 3.2.2.2), the term mediating coincides with the functionality of the mediator (3.2.2.3), and so on.

Klusch and Sycara [52] state that the semantics of terms matchmaker and broker as well as of mediator and broker are often used interchangeably in the literature; brokers are also often called facilitators.

The intention of all of the mentioned terms is to describe either the process of locating requested services and possibly managing the selection via a middle-agent or the same process without the middle-agent.

The diversity of dynamic architectures is very huge. The dynamic architectures start from simple a broadcasting method described in section 3.2.1, through many types of federated architectures, and to systems that use the acquaintance models of social knowledge described in section 3.2.3.

### 3.2.1 Broadcasting

The simplest architecture that offers the dynamic discovery of social knowledge uses the broadcasting technique. Whenever any agent needs to find an agent with a particular capability, the requester agent simply broadcasts a request to the whole community. Only those agents that have the requested capability should deal with this request and initiate a communication with the requester.

The broadcasting communication technique has to be supplemented by a negotiation protocol, i.e., the extensive and explicit use of communication to distribute the problem [138]. For instance, the Contract Net protocol can be used for
this purpose [120]. The agents (originators) in need of services distribute requests for proposals to other agents. The recipients of these messages evaluate those requests and submit bids back to the originating agents. The originators evaluate these bids, decide who is(are) the best agent(s) to supply the service, and then award the contracts to the selected agent(s).

The advantage of the broadcasting architecture is a simple implementation, since the implementation of special agents for negotiation is omitted. A second advantage is that there is not a single point of failure in the communication for this system. Another advantage is that the system that uses broadcasting is plug-and-play, i.e., open to new agents incoming to the system, terminating, or changing their capabilities.

A significant disadvantage in this system is that there is an overwhelming amount of communication in the community followed by the increased computation needed for message processing, because all agents in the system are forced to deal with every request. Apparently, the broadcasting architecture is not scalable. The requirement of this architecture is that the ability to broadcast messages has to be supported by any environment where agents are located.

3.2.2 Federated Architectures

The multi-agent system has a federated architecture if and only if the multi-agent system contains at least one special agent that provides social knowledge management. These special agents are usually referred to as middle-agents. When the middle-agents are present in the multi-agent system, then other agents can be referred as end-agents\(^2\) [141], but usually they are referred to as agents. Also note that an end-agent can also be a human in the case where the middle-agent is able to directly interface with a human\(^3\) [97]. There are many definitions of middle-agents.

- ‘Middle-agents are agents that help others to locate and connect to agent providers of services’ - Klusch and Sycara [52].
- ‘Agents that deal with preference or capability information that are neither requesters nor providers (from the standpoint of the transaction under consideration)’ - Decker et al. [15].
- ‘Middle-agent provides lookup services that facilitate the discovery of agents with specific capability descriptions; it may also mediate communication between agents.’ - Payne et al. [98].

We define a middle-agent as an agent that is able to handle providers offering information and requesters asking for information. The matchmaker (see

\(^2\) The term end-agent is not widely used though. Usually the term agent is used instead and a possible misunderstanding is eliminated by the context.

\(^3\) This is used very sporadically, e.g., in the A-Match system [97]. Usually the multi-agent system has a special agent that is used as an interface for a human.
section 3.2.2.1, broker (see section 3.2.2.2), and mediator (see section 3.2.2.3) are typical middle-agents. The terminology used in the literature is again not consistent since different authors either use the same term for different meaning or use different terms for the same meaning.

In the following sections we describe all types of federated architectures in detail. The goal is not only to describe the mechanism that is used by a particular federated architecture, but also to present advantages and disadvantages of the possible architectures.

### 3.2.2.1 Matchmaker

The Matchmaker approach is one of the federated architectures. The system consists of the providers and the requesters of information. The matchmaker agent mediates among the providers and the requesters. Figure 8 presents a simplified service matchmaking protocol that the matchmaker middle-agent uses to interact with providers and requesters. The simplified protocol depicts only those interactions that are a part of a successful negotiation and does not consider possible failures that can occur. Agent names consist only of the description of an agent class (see section 2.2.1.1), i.e., there can be multiple instances of an agent class and a message can be therefore sent to more than one instance of an agent.

![Figure 8: Service matchmaking](image)

1. **Registration phase** - Each provider registers its capabilities (services) with the matchmaker.

2. **Requesting phase** - The requester sends a request for a capability search to the matchmaker.

3. **Matching phase** - Whenever the matchmaker receives a request for capability from a requester, the matchmaker matches the request with current set of registered capabilities and returns a set of potential providers.

4. **Bidding request phase** - The requester contacts the providers, from the set of potential providers, with requests for service offers.
5. **Bidding reply** phase - Based on the service offer request, the providers should reply with their service offers - bids.

6. **Service request** phase - The requester chooses among the service offers and selects the best one according to the requester’s preferences. The requester then sends a request for the service to the selected provider.

7. **Service reply** phase - After the provider finishes the service, the provider sends the result of the service back to the requester.

The bidding phase (phases 4 and 5) is not required to be used in the service matchmaking protocol. If the requester does not want to choose among possible providers based on their offers, the requester can either blindly select one of the providers or choose the provider based on previous experience.

The service matchmaking technique is used in open systems for multi-agent system development since the matchmaker does not need to contain any application specific behavior. The matchmaker may be written as a general service that does not depend on the kinds of services and resources that are being matched [112], a definite advantage for matchmaking. One of the earliest matchmaker systems, based on the KQML specification (see section 4.1.1), is the ABSI (Agent-Based Software Interoperability) facilitator [119].

The matchmaker represents a smaller computation bottleneck than does the broker (see section 3.2.2.2), because once the match phase is completed, the matchmaker steps out of the scene. Nevertheless, the matchmaker cannot provide load balancing to optimize the usage of the providers.

The service matchmaking approach is also used in the coordination within the coalition formation methodology [117] and in the acquaintance model (see section 3.2.3) during an initial search for possible cooperators. The matchmaker then steps aside and is not involved in any further negotiation process.

The service matchmaking approach can be further improved by considering the track records of agents in accomplishing delegated tasks, as proposed in [147]. An assumption is that the requesters evaluate the providers after accomplishment of a task. These evaluations are then sent to the matchmaker, who stores them. This evaluation information can be used in the service matchmaking process to rank the providers.

### 3.2.2.2 Broker

The second well known type of federated architecture is the broker architecture [14]. Behavior of the broker architecture is similar to the service matchmaking approach (see section 3.2.2.1) at the beginning of a negotiation, but then differs significantly (see Figure 9).
1. **Registration** phase - Each provider registers its capabilities (services) with the broker.

2. **Requesting** phase - The requester sends a request for a capability search to the broker.

3. **Matching** phase - The broker matches the request with current set of registered services just as the matchmaker does. After the matching process, though, the broker behaves differently from the matchmaker because the broker does not return a set of potential providers to the requester.

4. **Bidding request** phase - The broker sends requests for the service offers to the providers.

5. **Bidding reply** phase - Based on the service offer request, the providers reply with their service offers to the broker.

6. **Service request** phase - The broker chooses among the offers and selects the best one according to the requester’s preferences. The broker then sends a request for the service to the selected provider.

7. **Service reply** phase - The provider sends the result back to the broker and then the broker sends the result back to the original requester.

The main advantage of service brokering is that the broker protects the privacy of the requester and provider. The requester and the provider each can not discover the identity of the other. Moreover, different types of encryption techniques can be used; the public/private-key encryption is the most suggested [27] and [140].

Another advantage of service brokering is that the broker can provide load balancing. The broker can choose providers not only to satisfy objectives of the requester, but also to optimize the usage of the providers [14].

Nevertheless, the broker has to understand preferences of the requester and also the meaning of these preferences to ensure that the broker chooses the best
provider. While using the service brokering mechanism, the broker represents a computation bottleneck in comparison with the service matchmaking mechanism. Also, both service matchmaking and service brokering can represent a single point of failure in the multi-agent system in the case where only one central middle-agent is used.

3.2.2.3 Mediator

The mediator architecture is based on the broker and matchmaker architectures with several changes. The mediator architecture originates from the usage of mediators in information systems [139], where ‘A mediator is a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications’. There are three layers within an application. The first layer consists of users, the second layer contains mediators, and the third layer is composed of databases.

Over and above the broker architecture, mediators provide learning and inconsistency checking. The mediator can provide advice on inconsistencies between acquired data and assumed knowledge and the mediator can learn by monitoring database changes and modifying certainty parameters in the knowledge base.

Usage of the mediators as a middle layer between the user’s workstations and data resources advanced to the point where mediators can be used in multi-agent systems and serve as advanced brokers [71]. The mediators in multi-agent systems also play the role of coordinators by promoting cooperation among agents and learning from their behavior.

As stated earlier, the mediator behavior draws its inspiration from both the matchmaker and broker architectures. The mediators can either use the service brokering mechanism to find proper agents as described in section 3.2.2.2, an example of an indirect collaboration, or use the service recruiting mechanism [58] (see Figure 10), a type of direct collaboration.

![Figure 10: Service recruiting](image-url)
The service recruiting mechanism is similar to the service brokering mechanism, except at the point where the mediator sends requests to the providers. The mediator leaves the negotiation at this point (in the same way as in the service matchmaking mechanism) and a communication is now directly among the requester and the providers. The requester decides which provider is the best one to be further contacted.

In the MetaMorph [72] implementation of the mediator approach, the mediators also provide task decomposition and dynamically form agent groups. The mediator decomposes high level tasks into lower level subtasks. Then all subtasks are subsequently handled by the mediator to determine the best solution. The mediator also uses the case based learning from previous agent interactions and learning from the future, where the mediator simulates the behavior of an environment and adapts to these predictions.

The mediator approach has the same advantages as do the matchmaker and broker approaches. In addition, mediators can decompose tasks, normally done by each agent separately, and can learn. Another advantage is the reduction of messages relative to the service matchmaking and brokering mechanisms. On the other hand, disadvantages include a higher computation load relative to the matchmaker and less autonomous behavior since mediators require application specific knowledge that normally is included only in the distributed agents.

### 3.2.2.4 Blackboard

The blackboard architecture can be used for dynamic social knowledge maintenance [15]. The blackboard architecture allows multiple independent agents to share information in a central store [62]. Therefore, the blackboard architecture can be used as a common place for social information storing and retrieval. The requesters post their problems on the blackboard and the providers can then query the blackboard for particular service requests. Other names for the blackboard architecture include newsgroups and bulletin boards.

The system consists of providers [125] or servers [14] that offer their services (capabilities), and of requesters of the services. The requester stores the request on the blackboard and in the meantime the provider periodically queries the blackboard for requests that the provider can solve. This is a synchronous type of blackboard (see Figure 11). If the provider finds such a request, the provider retrieves the request from the blackboard. After the result of the request is prepared, the provider stores the result on the blackboard. The requester in the meantime periodically queries the blackboard for the result and possibly retrieves the result when available.
Another type of blackboard architecture is an asynchronous blackboard, where a blackboard or a blackboard controller provides the ability to subscribe to particular events that may occur on the blackboard (see Figure 12).

For instance, the blackboard architecture that is used in the Hearsay-II project uses a precondition-action format in which the precondition determines when the action of a particular provider is applicable based on the current state of a blackboard [10]. Moreover, blackboards are typically applied to combinatorially explosive problems that are intractable and if the system attempts to execute all actions of the providers, an agenda-based control mechanism of the blackboard is used. All possible actions are placed on the agenda and rated. The most highly rated action is chosen for execution and removed from the agenda.

There is a communication bottleneck and single point of failure present in the blackboard architecture. However, one advantage is that it is an open architecture. The knowledge that any agent needs to access the blackboard architecture is the address of the blackboard and a communication language.

3.2.2.5 Monitor

The middle-agent that uses the content-based routing mechanism [58] is called a monitor [39] and [143]. The content-based routing mechanism is similar to
the service brokering mechanism (see section 3.2.2.2). The difference from service brokering is that the requesters subscribe to the middle-agent to obtain updates when a particular type of information changes. The providers, rather than advertising their capabilities, send updates to the middle-agent whenever there is a change in their knowledge base. The middle-agent then forwards these changes to all requesters that have subscribed to this type of information.

The advantage of the content-based routing mechanism is that the requester does not have to use communication with the middle-agent to search for a particular type of provider. The information about providers has already been gathered via the subscribe mechanism. Also the provider has to notify only the middle-agent to send updates to the requesters. Moreover, the privacy of both the requesters and the providers is increased.

The disadvantage of the content-based routing mechanism is that every change in the provider has to be propagated through the middle-agent to all subscribers, but there is no guarantee that all subscribers will be interested in this change. That means that a part of the communication and of the processing power is sacrificed during the update process in anticipation of benefit in the search process.

3.2.2.6 Facilitator

The facilitator approach is based on the grouping of agents, where one facilitator agent is assigned to one group. The facilitator agent then serves as a bridge for communication among agent groups [115]. The facilitator agent should not be misunderstood as the directory facilitator agent (see section 4.1.2). Systems that use facilitators are also called federated systems [30]. The facilitator approach is derived from the concept of a mediator [139].

For instance, the Open Agent Architecture (OAA), a framework for building multi-agent systems [71], has the facilitator as a main component. Not only is it used to route the messages among agents, but it also has an ability to serve as a broker. The OAA facilitator is able to use strategies and advice specified by the requester.

Communication among agents is possible only through the facilitators. The advantage of the facilitator approach is the reduction of communication channels in the system. Also the security of information can be increased in the facilitator approach. For instance the communication within one facilitator cannot be overheard by an agent that does not belong to this domain.

A similar architecture to the facilitator approach is the council approach [61]. Each fully interconnected group of agents contains one representative ‘Councilor’ that is similar to the facilitator. All councilors from all groups form a council that is also fully interconnected. The councilor approach that is called mediator councils is described in [32].

3.2.2.7 Embassy

The facilitator agent is very similar to the embassy agent [39] and [143]. The embassy agent may be requested to grant access to a domain. If the request is
granted, the embassy serves as a facilitator for a subsequent request. The embassy agent ensures the privacy of providers. Moreover, the embassy agent can translate incoming requests to the local language and vice versa.

### 3.2.2.8 Anonymizer

The anonymizer agent knows the preferences of a requester agent and its goal is to protect the requester and provider agents from learning agents [15]. Nevertheless, the cost for this protection is increased communication and increased possibility of a failure.

### 3.2.2.9 Job Agency

We propose the job agency architecture, one that offers a location of preferences and capabilities that is not known to be used so far in the area of multi-agent systems. In the job agency architecture, only a provider knows preferences; capabilities are known both to the provider and to the job agency. A requester does not have any preferences.

![Figure 13: Job agency mechanism](image)

1. **Registration** phase - Each provider registers its capabilities (services) with the job agency.
2. **Requesting** phase - The requester sends a request for a capability search to the job agency.
3. **Matching** phase - The job agency matches the request with a current set of registered services in a way similar to that of the matchmaker (see section 3.2.2.1).
4. **Job offering** phase - The job agency sends job offers to the providers that are known to have the requested capability. The providers use their preferences to accept or reject the offer.
5. **Job offering reply** phase - The providers that decided to accept the job offer reply back to the job agency.

6. **Service request** phase - The job agency chooses the first provider that replies to the job offer, because the job agency does not have any preferences to choose among the providers. The job agency then sends a request for the service to the selected provider.

7. **Service reply** phase - The provider replies back to the job agency and then the job agency forwards the result back to the original requester.

The job agency mechanism is similar to the service brokering mechanism (see section 3.2.2.2) in several aspects. First, both the job agency and the broker protect the privacy of the requester and the provider agents. Second, both middle-agents take care of the request from the initial request to the final solution.

The main difference between the job agency mechanism and the service brokering mechanism is that in the job agency architecture the requester does not have preferences by which the provider should actually solve the request. Also the providers do not express or expose their preferences to the middle-agent. The job agency chooses the first provider that replies to the offer and sends the request for service to this provider. Therefore, the job agency can act faster than the broker, because the broker has to wait for replies from all providers and then choose the best one. Nevertheless, the job agency does not perform a cost optimization, which is the main disadvantage.

### 3.2.3 Acquaintance Models

Assume that an agent is divided into two parts - an agent’s wrapper that usually deals with communication, and an agent’s body that contains the remaining activities of an agent. The acquaintance model contains social knowledge and is usually located in the wrapper of agents. The wrapper is split into a communication layer and acquaintance model [65]. Social knowledge is therefore distributed in a manner similar to that within the autonomous architecture (see section 3.1.2), but in this case the architecture is dynamic, i.e., can change over time.

Initially, at system startup, acquaintance models within all agents are either empty or they contain predefined information about other agents in the system. In the former case, any of the federated architectures can be used to obtain this information; usually the matchmaker architecture (see section 3.2.2.1) is used. Middle-agents handle the initial request for collaboration, and from this point the acquaintance model located in each agent keeps track of those agents in the system which are possible candidates for collaboration.

Two mechanisms can be used to keep the social information in the acquaintance model of an agent accurate. Periodic revision is the first possibility, where the agent periodically checks all cooperators for changes that occurred since the last update. A second possibility is to use a subscribe/advertise mechanism [65], where a subscriber is an agent that needs up-to-date information from cooperating
agents that, upon any significant change, send updated information to all subscribers.

The acquaintance model was initially based on a twin-base model consisting of knowledge about cooperators and tasks [8] (see [106] for the review of an evolution of acquaintance model architectures). Later a tri-base acquaintance model (3bA), an extension to ideas presented in the twin-based model, was introduced [66] and [100]. The 3bA model consists of the cooperator base, task base, and state base - each maintaining different types of information [106].

- **Cooperator base** - maintains permanent information on cooperating agents, for example name, address, and type of communication language.
- **Task base** - maintains semi-permanent knowledge about general problem solving, for example, possible plan decompositions and the actual state of planning processes.
- **State base** - maintains temporary knowledge about cooperating agents, for example, load, trust, and tasks currently being solved.

This differentiation provides an opportunity to split as much of the application specific knowledge away from the general mechanisms that can be used to efficiently maintain up-to-date social knowledge.

The key idea behind the acquaintance model architecture is that the communication that would be needed to obtain information about another agent is replaced by either periodic or change-triggered communication about agent state changes. The efficiency of the acquaintance model architecture depends on the number of agents, on the relations between agents, and on the number of dynamic changes of the state of agents that need to be transferred to the subscribers.

The advantage of the acquaintance model architecture is that the communication load is decreased at the time when the agent uses the knowledge stored in one of its knowledge bases. Nevertheless, the communication load is increased at the time of a knowledge update.

The disadvantage of the acquaintance model architecture is that there is no guarantee that the knowledge update will occur at the idle time when the agents are neither communicating to other agents nor internally processing a previous request.

### 3.2.4 Architecture without Middle-Agents

The architecture without middle-agents uses the technique for an agent location that does not contain middle-agents (introduced in [116]). This technique assumes that end-agents are able to obtain and store knowledge about other agents. Each agent $i$ holds a partial dynamically created list $L_i$ of other agents. The list contains at least the knowledge about agent names and their addresses. The size of $L_i$ has to be significantly smaller than the total number of agents in the system;
otherwise, the system would behave in a similar way as the broadcasting approach (see section 3.2.1).

Assume that all agents in the system already have their lists $L_i$ of agents created. The requester agent contacts all agents in $L_i$ to search for a service. If there is not any provider found for the requested service then the search continues one level further. A unique identification of the request prevents cycles in the search process.

The average depth of the search process in a planar rectangular lattice structure is close to the square root of the number of agents (see [116] for a detailed explanation). Further complexity reduction can be easily achieved by an introduction of shortcuts to the structure (see Figure 14), where the list $L_i$ of neighbor agents of any agent $i$ is indicated by a set of edges connected to the agent. For instance, neighbor agents of agent $A$ are marked by the rectangles in the Figure 14.

![Figure 14: Connectivity graph of a planar rectangular lattice structure with a shortcut](image)

The first shortcut introduced into the structure will on average reduce the search depth by 2.7%. Each additional shortcut provides a smaller reduction than the previous one. An example in [116] contains $10^4$ vertices in a 2-dimensional graph, resulting in an average search depth equal to 60. After adding 2000 shortcuts to the graph, the average search depth is reduced by 90% to 6, a very significant reduction of the search space, if we assume that the search method used is breadth-first.

Social knowledge in this architecture without middle-agents is partially static and partially dynamic. Interconnections among agents are created statically at design time. The interconnections count as social knowledge since information about addresses of neighboring agents is one of the basic examples of social knowledge. Nevertheless, the search process, which is used to locate providers for a particular capability, is a dynamic activity. Social knowledge is then also dynamically discovered via the search process.
The advantage of this architecture without middle-agent is that the system becomes more robust in comparison with federated architectures since social knowledge is not stored in a central place in the system, but obtained dynamically. Also, communication related to the distribution of knowledge among middle-agents is eliminated.

There are also many disadvantages of this architecture. To find all possible matches, the whole structure has to be traversed and all agents have to participate in the search process. Another disadvantage is that there has to be an entity in the system that takes care of creation and management of the whole connectivity graph. As all agents in the system are involved in the search process, the speed of the search can be significantly affected by the availability of agents. If there is a significant number of agents that are busy at the time of a search, then the search process has to wait for processing availability. Also, the possibility to introduce the privacy of a requester or of a provider into this type of system is very limited.

A similar approach designed and simulated in [90] uses a random search in the agent’s neighborhood followed by the grouping of successfully matched agents. Although the random search in the agent’s neighborhood has been designed to be ‘massively scalable’, the probability to find a successful match is around 95% (according to experiments presented in [90]). Lower than 100% success in matching, where we assume that there is at least one provider that offers the capability, can be a limiting factor in these systems.

The random search in an agent’s neighborhood provides search results in which any one of a number of possible matches is considered successful. This restriction means that there is no guarantee that all of the possible matches will be found. To use this technique there has to be a non-zero probability of finding neighbor agents located in the vicinity of a particular agent, which can also be a limiting factor.

A similar approach to the random search in an agent’s neighborhood is the clustering technique of agents used in a system called Yenta [26]. The main idea is again to group agents with similar interests. The agents clustering technique uses a more complex grouping than the agent’s neighborhood technique. The grouping is made bottom up and starts from grains of information that are grouped to collections called granules. The grouping of grains is called preclustering. When an agent finishes the preclustering phase of all local grains, then the clustering phase starts. During the clustering phase two agents compare their local granules by computing the highest similarity among all pairs of granules. Finally, the agents form a cluster in case that the highest similarity value is greater than a given threshold.

The advantage of the agents clustering technique is that the algorithm is dynamic, i.e., there are no predefined connections, and the algorithm is also decentralized. A disadvantage is that an agent with different interests than any other agent in the system will not appear in any cluster, which means that nobody will ever find this agent. Another disadvantage is that the computation complexity according to [26] is not better than the brute force technique.
Another clustering technique [142] uses a middle-agent\(^4\) to coordinate agents in the multi-agent system. The agents communicate using a fixed number of multicast addresses with a limited bandwidth and the goal is to group agents according to their similarity. The multi-agent system has one middle-agent at a time that has the role of active coordinator of the clustering process. The architecture uses the randomized timers election algorithm that ensures that when the middle-agent fails, a new middle-agent is elected from the remaining agents in the system. To decrease overloading during the election process, the agents start randomized timers only in the case where they succeed in an internal experiment with a random number that has to be greater than a given threshold.

3.2.5 Categorization of Dynamic Architectures

Decker et al. [15] categorize roles of middle-agents based on which type of social knowledge is known by the providers, requesters, and middle-agents. The capabilities can be known either by the provider only, or by the provider and middle-agent, or by the provider, middle-agent, and requester. On the other hand, the requester has preferences about the selection of providers. The preferences can be either private to the requester, or stored also in the middle-agent, or published so that the providers can also know them. The possibility to learn capabilities or preferences is not considered by this categorization.

The original categorization can be found, for example, in [15] and [52]. Nevertheless, we propose extensions to the list of middle-agents that belong to each category. Moreover, we generalized this categorization to all dynamic architectures, not only to middle-agents, and we added two new categories to the list. All of these changes are marked out by bold and cursive font in Table 1.

Decker et al. [15] state that ‘A specific request is an instance of an agent’s preferences, and a specific reply or action in service of a request is an instance of an agent’s capabilities’. The question is whether only the requester agent can have preferences. By close examination of the blackboard architecture, we found out that the preference does not originate only from the requester agent, but also from other entities. Therefore, we added new categories, where just the provider or provider and middle-agent initially know the preferences.

\(^4\) The middle-agent in [142] is named matchmaker, but it does not use the service matchmaking protocol. This type of middle-agent is an active coordinator instead of a passive matchmaker that is used to connect requesters with providers.
Table 1: Dynamic architectures categorized by initial privacy concerns

<table>
<thead>
<tr>
<th>Preferences initially known by</th>
<th>Capabilities initially known by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider</td>
<td>Provider + Middle</td>
</tr>
<tr>
<td>Requester</td>
<td>Broadcaster, <strong>Without Middle-agent</strong></td>
</tr>
<tr>
<td>Req. + Middle</td>
<td>Anonymizer</td>
</tr>
<tr>
<td>All</td>
<td>Blackboard</td>
</tr>
<tr>
<td><strong>Provider</strong></td>
<td><strong>Synchronous Blackboard</strong></td>
</tr>
<tr>
<td>Pr. + Middle</td>
<td>Asynchronous Blackboard</td>
</tr>
</tbody>
</table>

In the synchronous blackboard architecture (see section 3.2.2.4) a requestor stores the request for a capability first. A provider contacts the blackboard and searches for a request that can be solved by its capabilities. In this case, only the provider chooses which request to solve. After completion of the request, the provider stores the result on the blackboard. The requester can now retrieve the result from the blackboard, but neither the requester nor the blackboard can express any preferences. The privacy of the provider can be easily ensured in such a way that the capabilities of the provider are not revealed to the requester or to the blackboard.

In the asynchronous blackboard architecture a blackboard or blackboard controller provide the ability to subscribe to particular events that may occur on the blackboard. The providers store their preferences to the blackboard. In this case both the provider and the blackboard know the preferences. The requester agent contacts the blackboard to solve a request and the blackboard chooses the provider based on already stored preferences. After the completion of the request, the provider stores the result on the blackboard and the blackboard sends an event to the requester that the result has been obtained.

There is also the possibility that the provider in the asynchronous blackboard architecture wants to protect its capabilities. In this case the blackboard knows just the preferences and routes the requests without actually knowing the capabilities of the providers. Since the asynchronous blackboard architecture has these implementation options, it occupies two cells of Table 1.

Next we propose the job agency architecture (see section 3.2.2.9), where only the provider knows the preferences and capabilities are known to the provider and also to the middle-agent. In the job agency architecture the providers register their capabilities and preferences to the job agency first. Then the requester contacts...
the job agency middle-agent with the request. In the job agency architecture
the requester does not have preferences on which provider should actually solve
the request. The job agency contacts all providers that have the requested capability.
The provider chooses either to participate or not to participate based on its
preferences. The job solver chooses the first provider that replied to the offer and
sends request for service to this provider. After the completion of the request,
the provider sends the result back to the job agency. The job agency subsequently
forwards the result to the requester.

The matchmaker architecture (see section 3.2.2.1) can be used not only in
the case where the requester has preferences to choose from a list of suitable
providers, but also in the case where only the provider has the preferences.
The requester obtains the list of providers from the matchmaker and contacts all
providers from the list one by one. Each provider then uses its preferences and either
agrees or refuses to solve the request.

The architecture in which the capabilities of the providers are known to all
and the preferences are known both to the providers and to the middle-agent is not
a useful one. This architecture leads to the unnecessary revelation of information.
When the preferences are revealed by the provider to the middle-agent, it is not
necessary to reveal the information about the capabilities of the providers to
the requesters. Or, when the capabilities are revealed to the requester, then it is not
necessary to reveal the preferences of the providers to the middle-agent. Either
the matchmaker architecture or the asynchronous blackboard architecture can be
used instead.
4 Specific Management Techniques for Social Knowledge

Social knowledge has to be present in a multi-agent system to enable interoperation among agents, one of the key attributes of agents. Social knowledge is therefore also a part of the standardization process. The standardization process creates a foundation not only for the interoperation among agents within one multi-agent system, but also enables the interoperation among multiple systems.

The communication standards are also created with a consideration for social knowledge aspects. The standardization process is shaped by a practical utilization of social knowledge. Conversely, the standardization process also affects practical applications, since there is a tendency to use the standards in practice to accelerate and simplify the development process.

In the following sections, we present several standards that significantly affect social knowledge. Then we briefly describe several frameworks for the development of multi-agent systems and several types of multi-agent systems that use various social knowledge distribution and management techniques.

4.1 Social Knowledge in MAS Standards

Communication is one of the key areas in multi-agent systems. It is the glue among single agents that forms the whole multi-agent system. The role of communication is to ‘make the system temporarily and locally less distributed’ [70].

Communication can be split into two areas:

- **direct communication** - a communicative act with the purpose of transmitting information. Direct communication is aimed at a particular receiver or at multiple specified receivers.

- **indirect communication** - based on the observed behavior of other agents and on modifications of the environment. Indirect communication is usually a side effect of actions of an agent.

Social knowledge is used to find and select a receiver or receivers of particular information using direct communication. Social knowledge not only provides information about physical locations or addresses of other agents, but also about their capabilities, availability, trust, reliability, and so on. An agent uses social knowledge to select the best candidates among possible receivers in any particular situation.

Direct communication is an area in which the agent that transmits some information to other agents expects that the receivers will understand both the syntax
and semantics of the message (the meaning of the message and all of its parts). Direct communication is not just chaotic message passing. To get the meaning from a communication, there has to be a communication protocol that is used and understood by all agents that take part in the communication. Thus agents have to comply with communication standards to achieve the interoperability among agents within one multi-agent system and also across multi-agent system boundaries.

In the next sections, we briefly describe KQML and FIPA standards used in multi-agent systems. Then we focus on language specifications LARKS, SDL, and DAML-S used for social knowledge encoding. We also describe PHOSPHORUS and SLP service location mechanisms.

4.1.1 Knowledge Query and Manipulation Language (KQML)

KQML [22] is a language and associated protocol to support high level communication among agents. KQML is designed to offer a communication framework and any other communication language can be used as a content language. Moreover, the Secure KQML enhancement [111] has been defined to increase communication security.

KQML defines a set of performatives (message types) for communication between agents. From the point of view of social knowledge, KQML focuses also on the possibility of service matchmaking, brokering, and recruiting (see section 3.2). Matchmakers can be asked to ‘recommend-one’ and ‘recommend-all’, brokers to ‘broker-one’ and ‘broker-all’, and mediators to ‘recruit-one’ and ‘recruit-all’ [21] with performatives ‘tell’, ‘advertise’, and ‘subscribe’.

The examples of service matchmaking and service brokering can be easily handled by described performatives [67] (see Figure 15 and Figure 16).

![Figure 15: Service matchmaking in the KQML example](image)

In the service matchmaking example the provider advertises a capability to answer to requests for the value of $X$. When the requester needs the value of $X$, the requester sends the performative `recommend` with the parameter `ask(X)`. The matchmaker then replies with a set of providers and the requester can then use `ask(X)` to send the request directly to the chosen provider. Finally, the provider uses `tell` to send the result back to the requester.
In the service brokering example, the requester, instead of sending the performative `recommend`, sends the performative `broker` with `ask(X)` as a parameter. The broker then selects the best provider of this request and asks the provider about the value of X. When the provider `tells` the result, the broker `tells` this result back to the requester. Note that the bidding phase (see section 3.2.2) is for simplicity not considered in this case.

### 4.1.2 Foundation for Intelligent Physical Agents (FIPA)

FIPA [24] is an international non-profit association of companies and organizations sharing the effort to produce specifications of generic agent technologies. FIPA is becoming a widely used standard for the multi-agent system development.

One of the FIPA specifications is the Agent Communication Language (ACL). FIPA ACL specifies a standard message language by setting out encoding, semantics, and pragmatics of the messages. FIPA does not specify a mechanism for transportation of messages within an agent platform, only declares that messages should be encoded in a textual form to overcome bridging into heterogeneous platforms.

FIPA specifies three mandatory agents that have to be present in any multi-agent system:

- The Agent Management System (AMS) - provides the authentication of resident agents and the control of registration services (‘white pages’).

- The Directory Facilitator (DF) - offers agent capabilities registering and lookup services (‘yellow pages’). The behavior of the DF agent is similar to the service matchmaking mechanism (see section 3.2.2.1).

- The Agent Communication Channel (ACC) - provides message routing (bridging) within or over the borders of an agent platform. The behavior of the ACC agent is similar to the embassy agent (see section 3.2.2.7).

The DF agent is the most important from the point of view of social knowledge. FIPA specifies that at least one DF agent has to be resident on each Agent Platform (AP) [23]. However, the AP may support any number of DF agents.
In this case the DF agents can register with each other. Also, other mandatory agents (the AMS and ACC) can register with the DF agent.

The DF agent has to support the following management actions:

- **Register** - used by any agent to register its services to make them public.
- **Search** - request information about service providers.
- **Deregister** - used by an agent to remove its information.
- **Modify** - used by an agent to modify its registration information.

The search request to the DF agent is handled by a search mechanism. First, the DF searches locally. If the result is not sufficient, the DF triggers a search to the other DF agents in the hierarchy. There are four attributes that can be specified for each search.

- Maximal *number of responses* to be found.
- Search *algorithm* is depth-first by default, but any other algorithm can be used when properly advertised.
- Search *filter* to specify domain boundary for searching
- Search *time & duration* to limit process of searching by absolute or relative constraint.

FIPA does not specify communication between the DF agents to resolve search requests that are deeper than one level. Also, FIPA does not specify how the DF agents should be distributed in the system.

### 4.1.3 Language for Advertisement and Request for Knowledge Sharing (LARKS)

LARKS ([123], [124], and [125]) has been designed for multi-agent systems that use the service matchmaking mechanism (see section 3.2.2.1). LARKS focuses more on an agent capability description instead of message types. LARKS specifies the *context* of a message, the definition of used *data types*, *input* and *output variables*, *input* and *output constraints*, *description* of the meaning of words, and an optional *description*. All of this information is entered as text strings.

An assumption for an efficient LARKS utilization is that provider and requester agents in open systems do not have to share the meaning of used words, i.e., a common ontology in the open system does not have to exist.

The following matching techniques are used during the processing of requests in the Matchmaker:

- The *exact match* succeeds if and only if the request and the stored knowledge are exactly the same.
- The plug-in match, or logical match defined in [46], succeeds, when the request is semantically a subset of stored knowledge. Usually an ontology is used to compute the semantic or subsumption matching.

- The relaxed match, or relaxed exact match defined in [46], computes syntactic and semantic distances between the request and the stored knowledge and succeeds if the distance is no higher than a given threshold. The word distance is computed based on the Wall Street Journal corpus of one million word pairs in a current implementation [124]. For the most similar words that are found during the comparison the semantic distance is also computed to ensure that the attached concepts are also similar. The GRAPPA matchmaking framework uses a similar matching technique called multidimensional matchmaking, but instead of the word distance the built-in distance functions are used to compute the distance that is used to rate the match.

The plug-in match and the relaxed match can be efficiently used in open systems where providers offer something slightly different than requesters ask for. If they do not use languages that are too different then the matchmaker should be able to find capabilities corresponding to the request.

4.1.4 Service Description Language (SDL)

SDL [122] offers a similar type of specification as does LARKS (see section 4.1.3). The description of an agent service consists of the following components:

- service name - an expression that describes the service;
- inputs - a set of mandatory inputs and a set of optional inputs that are sent from the requester to the provider;
- outputs - a set of mandatory and optional outputs that will be part of the reply from the provider; and
- attributes - an additional set of optional attributes, for instance, cost, response time, or evaluation rating of the service.

4.1.5 PHOSPHORUS

The PHOSPHORUS agent matchmaking service [34] uses several types of reasoning that are based on definitions of the domain terms and ontologies that are a part of the domain knowledge bases.

The semantic agent capability matching can use, according to the classification proposed in [33], four types of semantic matching:

- Subsumption-based match - the request is subsumed by a capability of an agent. This is the same mechanism as the plug-in match (see section 4.1.3).
- **Reformulation-based match** - the request can be satisfied by combining the capabilities of two or more agents. This type of match is a form of divide-and-conquer. Nevertheless, the Contract Net protocol [120] is used currently for the same purpose and is implemented by end-agents instead of by middle-agents, as the reformulation-based match is.

- **Reverse subsumption-based match** - only a part of the request can be satisfied by an agent. There is no guarantee that the resulting agent will be able to solve the original request.

- **Partial match** - an agent has a capability that is similar or related to the original request. The partial match also does not guarantee that the resulting agent will be able to solve the original request.

### 4.1.6 DAML-S

The service description language DAML-S provides a semantically based view of services which extend from the abstract description of the capabilities of the service to the specification of the service interaction protocol to the actual messages that it exchanges with other web services [95] and [96].

The DAML-S language is, in concept, similar to LARKS (see section 4.1.3). The capability matching is based on service profiles that describe the functionality that a web service wants to provide to a community. The service profile describes the capabilities of a service in terms of inputs, outputs, preconditions, and effects. Both the service requests and the services provided are specified by the service profiles.

The service-matching algorithm *flexible matching* is based on the exact match and the plug-in match, which are ported from LARKS to achieve the same results. To match the request, the flexible matching algorithm loops through all advertisements and scores them to find the matches with the highest rating.

A match is recognized if and only if for each output of the request, there is a matching output in the advertisement. The same algorithm computes the matching between inputs, but with the order of the request and advertisement reversed, i.e., the advertisement’s inputs are matched against the requester’s inputs.

The rating of the match between two inputs or two outputs depends on a relation between associated concepts. The flexible matching algorithm computes a subsumption relation by the minimal distance between concepts in a taxonomy tree (or ontology).

### 4.1.7 Service Location Protocol (SLP)

SLP [36] is an Internet Engineering Task Force standard for enabling network-based applications to automatically discover the location of a required service, including an address or a domain name and other configuration information. SLP provides fully decentralized operation and scales from small networks that are
not administered to large enterprise networks with policies dictating who can discover which services.

SLP defines three types of agents [36]:

- User Agents (UA) (the requester agents) - perform service discovery on behalf of client software.
- Service Agents (SA) (the provider agents) - advertise the location and attributes on behalf of services.
- Directory Agents (DA) (the middle-agents) - aggregate service information into a repository.

SLP uses two types of service discovery - *active* and *passive*. In the active discovery user agents and service agents broadcast requests over a network to locate directory agents. In the passive discovery directory agents broadcast advertisements for their services and continue to do this periodically in case other agents failed to receive the initial advertisement.

It is also possible to use SLP without middle-agents. In such case, the requester agent repeatedly broadcasts the same request that would be sent to the middle-agents. Provider agents listen for the broadcasted requests and then they send responses back to the requester.

Service registrations have lifetimes no longer than 18 hours. Therefore, provider agents have to reregister the service periodically before the lifetime expires.

SLP supports multiple middle-agents distributed over the network. Distributed middle-agents are duplicate repositories of service information without requiring any formal database synchronization among them. Because requesters can choose to contact any available middle-agent to issue the request, the load will be shared among the middle-agents and the middle-agents are no longer a single point of failure.

To increase the scalability of the SLP architecture, middle-agents can be used to group resources by location, network, or administrative category. Requesters and providers are initially configured to use the scope string ‘default’. An administrator can control the set of services available to a particular client by the scope strings to ensure either the security of particular resources or to impose restrictions on particular requesters.

### 4.2 Social Knowledge Management in MAS Implementations

There are hundreds of different implementations of multi-agent systems so far. To summarize them all is beyond the scope of this work. Instead, we described several of the most used open multi-agent system development environments.

JADE and FIPA-OS are implemented in the JAVA programming language and they are available as open source software. Both of them are FIPA-compliant.
(see section 4.1.2). The FIPA-compliant systems are typically federated architectures since FIPA specifies that both AMS and DF mandatory platform agents have to be present in the system [24] (see section 4.1.2).

The ProPlanT system is one of the systems in which implementation is based on the acquaintance models of social knowledge (see section 3.2.3) embedded in each agent [65]. SHADE and COINS are KQML based systems that support many types of middle-agents. The InfoSleuth system is focused on information discovery and retrieval in a dynamic and open environment.

4.2.1 Java Agent DEvelopment Framework (JADE)

The JADE is a software framework that eases the development of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems [5] and [44]. The goal of JADE is to ease development while ensuring standard compliance through a comprehensive set of system services and agents.

JADE offers the FIPA-compliant Agent Platform (AP), including the AMS, ACC, and DF mandatory system agents. All three agents are automatically activated at AP startup. JADE supports multiple domains and FIPA-compliant DF agents can be started in these domains at run-time and linked in a federation, thereby implementing a multiple domain agent environment. JADE offers a graphical user interface for creation of domain hierarchies with multiple federated DF agents.

The DF uses standard ACL communication and can be contacted to register, deregister, modify, and search. The DF could contain any of the following agent attributes: an address, service, interaction protocols, ontology, and ownership. The search method is based on constraints, where any attribute can be used to specify the search. More DF agents can be used to divide the AP into many Agent Domains. Moreover, version 2.2 of JADE supports and guides creation of the DF federations to build the hierarchical structure of the DF agents.

4.2.2 FIPA Open Source (FIPA-OS)

FIPA-OS is a component-based toolkit enabling the rapid development of FIPA-compliant agents [25]. FIPA-OS supports the majority of the FIPA experimental specifications and is being continuously improved as a managed Open Source project.

FIPA-OS offers the FIPA-compliant AP, including the AMS, ACC, and default DF mandatory system agents (see section 4.1.2). FIPA-OS supports different types of Agent Shells, all of which can produce agents that can communicate using the FIPA-OS facilities [107].

Agents who offer services first contact the DF agent. At a later time requester agents contact the DF by creating tasks to search for a particular service. These tasks use a DFSearchDescription structure as a search constraint. The DF returns a DFSearchResult structure containing an array of DFAgentDescription with
information about each found agent. FIPA-OS does not specify any more complex directory facilitator structures.

### 4.2.3 Production Planning Tool (ProPlanT)

ProPlanT ([66] and [110]) is an implementation of a multi-agent system based on the acquaintance models of social knowledge (see section 3.2.3). The architecture of agents is fully hierarchical with three types of layers. A project-planning layer is the topmost, a project-managing layer is in the middle and can be internally interconnected, and a production layer contains agents with direct connection to production processes.

All agents in this system are connected peer-to-peer and social knowledge is incorporated into the wrapper of each agent and thus each agent is fully autonomous (see section 3.2.3). Social knowledge in the running system is both maintained autonomously by each agent and contained by the meta-agent [101] (see section 6.3.2).

The meta-agent does not have any brokering or matchmaking capability, but the meta-agent observes communication among agents. The meta-agent is able to affect acquaintance models of agents using revision mechanisms suggesting changes based on changes in the system [67] and in some sense interacts in a way similar to the monitor agent [39]. The main advantage of this approach is that meta-agents do not burden the community in critical communication points, but they wait with suggestions until moments of low communication load.

### 4.2.4 SHAred Dependency Engineering (SHADE) and COmmon INterest Seeker (COINS)

Both SHADE and COINS architectures ([78] and [58]) are based on KQML communication (see section 4.1.1). Moreover, a major part of the SHADE effort is an ongoing contribution to KQML. Both architectures are based on matchmaking provided by middle-agents.

SHADE and COINS middle-agents support four types of matchmaking using the following KQML performatives:

- `recommend` - the service matchmaking (see section 3.2.2.1)
- `broker` - the service brokering (see section 3.2.2.2)
- `recruit` - the service recruiting (see section 3.2.2.3)
- `tell` - the content-based routing (see section 3.2.2.5), where ‘tells’ from providers are forwarded to all requesters that have subscribed to the middle-agent.

---

5 The term matchmaker is used in [78] and [58]. To be consistent with the terminology, we use the term middle-agent because it better reflects its functionality.
The matchmaking algorithm of the COINS middle-agent utilizes the TF-IDF (Term Frequency-Inverse Document Frequency) mechanism [78] to compute relaxed match distances. The SHADE middle-agent uses the unification process instead.

4.2.5 InfoSleuth

InfoSleuth and its successor, InfoSleuth II, are agent-based systems for information discovery and retrieval in a dynamic and open environment ([84], [85], and [4]).

InfoSleuth’s middle-agents offer not only syntactic matchmaking, but also the semantic matchmaking functions to complement the syntactic matchmaking process. The semantic matching process is based on the common service ontology, an approach that is similar to LARKS (see section 4.1.3) and DAML-S (see section 4.1.6).

The architecture of InfoSleuth allows using either a single middle-agent or multiple ones. Matchmaking with multiple middle-agents uses peer-to-peer topology for communication among middle-agents, a similar topology as the one that is used by the teamwork-based technique (see section 5.3.2). The difference is that in the peer-to-peer topology the middle-agents may specialize to specific domains. An agent has to ensure an advertisement of its capabilities to middle-agents that best represent its interests. The system should also contain at least one general purpose middle-agent to handle queries that are not handled by the specialized middle-agents.

---

6 The term brokering is used in [84]. To be consistent with the terminology, we use the term matchmaking.
5 Social Knowledge Distribution

By stepping out of the area of multi-agent systems and looking at social knowledge distribution and management in real life, it is clearly visible that social knowledge is distributed various ways and it can also be searched various ways. For example, assume that David is searching for a carpenter. First of all, David could already know a carpenter. Or, he can possibly ask his relatives that live nearby or neighbors whether they know a carpenter. Or, he can open a yellow pages book that possibly contains a list of carpenters that work in his region. Or, he can connect to the Internet and search the digital yellow pages for information about carpenters that live within his state. Once David obtains the desired knowledge, he probably remembers it for some amount of time, i.e., he becomes a knowledge source about carpenters for himself and also for others, and thus the social knowledge becomes even more distributed than before.

All of these sources, plus others, can be used to obtain desired knowledge. Therefore, the question is: why not have one central social knowledge source for the whole world? A problem arises when this one central source is unavailable. David must wait until he accidentally meets a carpenter or becomes a carpenter himself in this case. Therefore, the central source must be at least backed up or duplicated. To insure that an earthquake cannot destroy all knowledge sources, they have to be physically distributed. Imagine that David’s house suffers a loss of power. How he can access the central social knowledge source to search for an electrician? A yellow pages book could be a solution for this situation, but this book is again another type of distributed social knowledge.

In a multi-agent system social knowledge can also be distributed in many different ways based on the system architecture as specified in section 2. First, section 5.1 focuses on architectures in which social knowledge is centralized, but we show possibilities for enhancing them to overcome a communication bottleneck and single point of failure (i.e., a central element whose possible failure affects the whole system). We describe in section 5.2 architectures in which social knowledge is distributed and in section 5.3 architectures that are hybrid.

5.1 Centralized Social Knowledge

When social knowledge is centralized, it is stored, managed, and offered from one physical or functional point. In this section, we describe multi-agent system architectures where social knowledge is centralized. First, we examine the blackboard architecture as a typical representative of centralized architectures. Then we focus on federated architectures in which only one middle-agent is used to deal with social knowledge.
5.1.1 Blackboard Architecture

A typical representative of architectures in which social knowledge is centralized is the blackboard architecture (see section 3.2.2.4). This architecture is typically based on a central point in the system where all contributors meet. The blackboard architecture has been designed to manage knowledge in general and therefore this architecture can also be used for social knowledge in a multi-agent system.

The blackboard has an advantage in its very efficient centralized control mechanisms. The design and implementation of distributed control mechanisms can be very time consuming and can even decrease overall system efficiency at run-time. Usually, scalability and robustness are not required for applications that are built using the blackboard architecture. Nevertheless, the blackboard architecture can be extended to become distributed (see section 5.3.4).

5.1.2 Federated Architectures

If the multi-agent system contains at least one special agent that provides social knowledge management, we are talking about a federated architecture (see section 3.2.2). All types of federated architectures are in their simplest form centralized. If the multi-agent system contains only one middle-agent, social knowledge is centralized within this middle-agent. This type of system is described in [32] as a single mediator network.

As multi-agent systems grows in size and complexity the centralized model becomes a bottleneck and single point of failure of the architecture since all social knowledge traffic goes only through this one middle-agent. Many descriptions of the federated architectures mention that it is possible to have more than one middle-agent in the system, but they stop at this point and do not specify how this system might work. For instance, [58] anticipates that techniques to distribute the matchmaker load will be required as the application grows in size and complexity.

5.2 Distributed Social Knowledge

To avoid a communication bottleneck and single point of failure in the multi-agent system, it is essential to distribute social knowledge. Any type of static architecture described in section 3.1 has social knowledge statically plugged directly into all agents at design time. But the static approach makes the whole system closed and inflexible. How the dynamic architectures can support distributed social knowledge is described in the following sections.

5.2.1 Acquaintance Models

The architecture that uses acquaintance models (see section 3.2.3) is an example of a dynamic architecture in which social knowledge is distributed among all agents. Middle-agents can be used in this approach, but they provide...
services only for system initialization and the initialization of a new agent entering a running system.

The advantage of the acquaintance models is that there is no communication bottleneck in the system. Also, any failure of an agent has a minimal impact on system functionality since social knowledge is distributed among the agents. Agents communicate peer-to-peer and for social knowledge management they use the subscribe/advertise mechanism (see section 3.2.3).

5.2.2 Architecture without Middle-Agent

Another example of distributed social knowledge in a dynamic architecture is the architecture without any middle-agent (see section 3.2.4). Social knowledge is gathered from agents at the time when the requester searches for this knowledge.

The key to this approach is in a structure of interconnections among the agents. The search for information is managed by a series of small broadcasts that can be heard by neighboring agents, where the interconnections among agents are determined at design time and some of them can interconnect physically distant agents to provide shortcuts for the search process.

This architecture, with no middle-agent, uses the statically predefined structure of interconnections among agents to dynamically search for social knowledge within the distributed community of agents. Social knowledge is therefore distributed, but without a middle-agent.

5.3 Hybrid Distribution of Social Knowledge

We examine several architectures that are hybrid in distribution of social knowledge, i.e., where social knowledge is organized into groups, hierarchies, etc.

5.3.1 Facilitators

The facilitator architecture (see section 3.2.2.6) is based on a grouping of agents, where one facilitator agent serves as a bridge among agent groups. We are also assuming that facilitators are the sources of social knowledge within their respective domains. The facilitator architecture lies in the middle between centralized and fully distributed approaches. The first extreme occurs when we define just one group for the entire system -- the system becomes centralized. The second extreme occurs when we define only one agent per group. In this case, the system becomes fully distributed.

5.3.2 Teamwork-based Technique

The facilitator architecture described above, further improved in [55] and [56], is called a teamwork-based technique. The structure of the middle-agents is very similar to the structure of facilitators (see section 3.2.2.6), but instead of facilitators the teamwork-based technique uses middle-agents. The teamwork-based
technique also lies in the middle between centralized and fully distributed approaches. The main characteristic of the teamwork-based technique is that all middle-agents are sources of social knowledge about all domains. Social knowledge is not only distributed into a particular number of domains, but also replicated the same number of times.

There is one assumption in the teamwork-based technique, namely, that all middle-agents in the system have to be interconnected (the same as the facilitator architecture). This architecture is described in [32] as a complete network of mediators.

The advantage of the teamwork-based technique is that the system is very fault-tolerant to the unavailability of any middle-agent. To be precise, the system is fault-tolerant to $N-1$ failures of $N$ middle-agents, and thus the maximal failure impact $FI_{N-1} = 0\%$ (see definition in section 7.3).

The teamwork-based technique needs more resources such as memory and processing power, which is a disadvantage. Since the same information is replicated $N$ times in the system, there is also the possibility that an inconsistency can appear. Also, the architecture does not scale well since all middle-agents have to be interconnected, resulting in $N(N-1)/2$ connections.

5.3.3 Distributed Matchmaking

The distributed matchmaking agent architecture (DMAA), proposed in [109] and currently in development, has a different goal than the teamwork-based technique (see section 5.3.2). The goal of DMAA is not to increase the robustness of the middle-agent architecture, but rather to decrease the system workload by reducing the communication between middle-agents. The DMAA consists of three types of middle-agents:

- local matchmaking agent (LMMA) - responsible for local requests and advertisements.
- authoritative matchmaking agent (AMMA) - domain specific matchmaker that is used when the local matchmaking agent was unable to find requesting providers.
- root matchmaking agent (RMMA) - exploits a taxonomic knowledge base to perform an ontological load balancing among AMMA agents in the system.

When the LMMA is not able to find requesting providers, the RMMA is used to recommend one of the authoritative matchmaking agents that is the most suitable to help based on the taxonomy located in the RMMA.

The advantage of this architecture is that the throughput of the matchmaking services of the system is increased in comparison with the single matchmaker system. A disadvantage is that the RMMA is a single point of failure and also a communication bottleneck for all non-local requests.
5.3.4 Distributed Blackboards

One possible way to make the blackboard architecture (see section 3.2.2.4) distributed is to split the blackboard into several more specific blackboards. Requesters store their requests on blackboards that are related to the type of request, and providers contact only those blackboards storing relevant requests. The connection among blackboards, requesters, and providers can be based on periodic queries or be event driven using the subscribe/advertise mechanism.

For instance, the blackboard architecture called CASSANDRA [10] consists of level managers, where each level manager includes its own set of providers, its own local database of partial solutions, and its own local control mechanism. The level managers are interconnected via communication channels that provide the only way for information exchange.

Another approach to distribute the centralized blackboard architecture is described in [62]. This distributed blackboard architecture consists of two types of blackboards that can be used to create two levels of hierarchy in the blackboard architecture: the local blackboard that resides in each host in the distributed environment and the global name manager that maintains the information of all local blackboards in the system.
6 Social Knowledge Robustness

Most of the theoretical research in the area of multi-agent systems deals with design, creation, and management of the system. The research usually stops at the point when the system works according to the desired behavior. But another very important aspect affecting the deployment of a system is its robustness. It has been noted [47] that there is increasing interest within multi-agent system research in the ability to provide robust and efficient systems. One of the main advantages of the multi-agent system should be that it offers a robust and survivable system.

The robustness in multi-agent systems can be addressed from two points of view:

- First, from the point of view of the whole system; ‘When information and control is distributed, the system is able to degrade gracefully even when some of the agents are out of service temporarily’ [128].
- Second, from the point of view of an agent; ‘Robust behavior in complex, dynamic environments mandates that intelligent agents autonomously monitor their own run-time behavior, detect and diagnose failures, and attempt recovery’ [49].

The definition of robustness in the area of software testing is: ‘The degree to which the system or component can function correctly in the presence of invalid or conflicting inputs’ [43].

6.1 Fault Tolerance Degrees

One advantage of a multi-agent system is the robustness derived from its distribution of knowledge and control. A multi-agent system consists of a set of agents that participate in the functionality of the system. When an agent fails (see section 7.1), the impact on the whole system should not be larger than the loss of the corresponding functionality embedded in the failed agent.

Since we are dealing with social knowledge management in the multi-agent system, we focus on failures there. We described several types of social knowledge distribution in section 4 - centralized, distributed, and hybrid. The failure impact on the whole system functionality is related to the type of architecture that is used.

The impact of the failure on the system is usually not easy to anticipate, and also not easy to measure. There are three levels or degrees of fault tolerance [37] already defined for multi-agent systems.

- Full fault tolerance, where the system continues to operate without a significant loss of functionality or performance even in the presence of faults.
• **Graceful degradation**, where the system maintains operation with some loss of functionality or performance.

• **Fail-safe**, where vital functions are preserved while others may fail.

  We can add a *fail-unsafe* system, where the system is unable to preserve its vital functions.

  These three or four degrees of fault tolerance provide only very rough measure. Moreover, we usually cannot determine which degree of fault tolerance will occur after a particular failure. For instance, assume that one of the middle-agents stops responding and the knowledge that this middle-agent holds is not redundant in the system. In this case we cannot distinguish whether this loss of knowledge will affect vital functions of the system, the functionality or performance of the system, or whether the system can continue to operate without a significant loss of functionality or performance.

### 6.2 Vulnerability Measure

A measure more precise than these degrees of fault tolerance is the vulnerability measure used in [15]. Vulnerability is defined as a sum of expected costs of each possible failure multiplied by the probability of that failure. The main problem that arises in a practical application is that it is usually not possible to obtain the probabilities of all possible failures of all types of agents in the system. Another disadvantage of the vulnerability measure is the impossibility of measuring an impact on the system when more than one failure occurs in the system simultaneously.

To assure that the system will be able to degrade gracefully and not stop functioning when a failure occurs, we now describe how to increase robustness.

### 6.3 Techniques to Increase Robustness

Several types of techniques can be used to increase the robustness of either one single agent or the whole multi-agent system. First, traditional fault-tolerance techniques as warm/hot backup (see section 6.3.1.1) or N-version programming (see section 6.3.1.2) can also be applied to the area of multi-agent systems. Second, several types of monitoring techniques have been developed to increase the robustness of the whole multi-agent system (see section 6.3.2).

#### 6.3.1 Traditional Fault-Tolerance Techniques

The traditional fault-tolerance techniques have not been developed to be specifically used in multi-agent systems. The development of these techniques is targeted to database systems, application servers, resource managers, and distributed systems, but these techniques can be used in a much broader range of software or hardware systems. The summary of these techniques is described for instance in [56]. We will focus only on techniques that can be directly applied to multi-agent
systems that use middle-agents for social knowledge management, in the event that a middle-agent fails.

6.3.1.1 Warm and Hot Backup

Assume that the multi-agent system has one primary middle-agent in an active state and one secondary middle-agent in an inactive state. In the active state, a middle-agent can be contacted by other agents and can respond to their requests for services. While in the inactive state, a middle-agent can not be contacted.

In the warm backup technique, the initial goal of the secondary is only to observe the primary and wait for a failure of the primary. When the failure occurs, the secondary becomes active and starts recovering to the last known state of the primary. After the recovery process has been completed the secondary becomes the primary middle-agent.

In the hot backup technique, the initial goal of the secondary middle-agent is not only to observe the primary and wait for a failure of the primary, but also to monitor inputs and outputs of the primary. When the failure of the primary middle-agent occurs, the secondary middle-agent can immediately take over as primary without the recovery phase.

6.3.1.2 N-Version Programming

N-version programming is defined as an independent generation of \( N \geq 2 \) functionally equivalent programs, called ‘versions’, from the same initial specification [12].

The ‘independent generation of programs’ means that the programming efforts are carried out by \( N \) individuals or groups that do not interact with each other relative to the programming process. Whenever possible, different algorithms, programming languages, and translators are used in each effort.

All \( N \) versions are running simultaneously and a voting mechanism is used to compute the result. Either exact voting, where all versions have the same number of ballots, or inexact voting, where each version has a specific number of ballots, can be used as the voting mechanism.

A fault-tolerant pipeline technique [80] is an example of \( N \)-version programming in multi-agent systems, in which a fault-tolerant computation is achieved by multiple computations of the same problem running simultaneously. Several agents are used for the same phase of the computation and each partial result is obtained by the voting process.

Middle-agents can use \( N \)-version programming in two ways:

1. A middle-agent consists of \( N \) modules that are developed via \( N \)-version programming. All \( N \) modules run simultaneously to compute results and the final result is selected by the voting process.
2. The multi-agent system has at least $N$ middle-agents that are developed via the $N$-version programming. The middle-agents do not have to necessarily vote. The presence of $N$ differently developed middle-agents in the multi-agent system increases the robustness.

6.3.2 Multi-Agent System Monitoring Techniques

The Meta-Agent [101] monitors communication among agents and gives guidelines to increase reliability of the multi-agent system and can also be used to detect failures in the community of agents (see section 8.1.6.3). The meta-agent has either a passive role in the community and provides the information about how the community evolves in time or an active role in which the meta-agent affects directly other agents within the community.

The development of Socially Attentive Monitoring (SAM) [49] focuses on a plan recognition, where an agent can infer other socially similar agents’ beliefs, goals, and plans from their observable actions and their communication, thus possibly detecting a failure. The SAM is composed of three phases; a failure detection phase, social diagnosis phase, and a recovery phase that results in the increased robustness of the whole system.

The Sentinel [37] is a special agent and its goal is to guard specific functions of a multi-agent system or to guard against specific states of a multi-agent system. The sentinel interacts with other agents either by monitoring communication among agents or by requesting some additional information. The sentinel can use timers to detect possible no-response failures (to be defined below). As the sentinel builds internal models of other agents, it is able to detect a failure not only from behavior of the agent, but also from its internal state and can lead to early detection of a failure. The multi-agent system can contain more than one sentinel in its environment to either increase the robustness or to provide better scalability.
7 New Classification of Failures in MAS and Failure Impact

Different types of failures can occur in agents and in multi-agent systems. Each one has an impact on the robustness and survivability of the system architecture or on the particular implementation of the system. Thus we define in section 7.1 possible types of agent failures and, in section 7.2, possible types of multi-agent system failures. In section 7.3 we propose middle-agent failure impact attributes that can be used to assess the static impact of such a failure on the whole multi-agent system. We also define average redundancy and weighted average redundancy measures that are not dependent on the number of failures that occur simultaneously. Finally, in section 7.4, we describe how graph theory deals with fault tolerance and we show that this approach can be used to evaluate the dynamic impact of middle-agent failure in a multi-agent system.

7.1 Agent Failure Types

There are numerous reasons why a particular agent provides incorrect information or fails to behave correctly. To recognize these failures, we suggest a classification of agent failures into the following categories (see Figure 17).

Bold lines in Figure 17 represent a subclassing. The words in italics complemented by dashed lines, pointing to a particular type of failure, represent reasons that can lead to that particular type of failure.
First of all, we provide definitions for terms that will be used in the following descriptions, taken from the area of software testing [51].

- **Mistake** - A human action that produces an incorrect result.

- **Fault** - An incorrect step, process, or data definition in a computer program. The outgrowth of the mistake. (Potentially leads to a failure.)

- **Failure** - An incorrect result. The result (manifestation) of the fault (e.g. a crash).

- **Error** - The amount by which the result is incorrect.

From these definitions we can conclude that:

- A fault is time invariant. The fault manifests when it is used at run-time, for instance, when the user tries to compile an executable, to run the executable, to test its response to particular inputs, and so on.

- A failure is always the manifestation of at least one fault, although it can be sometimes very hard to point at the fault itself.
Failures of an agent can be further classified by their observability (see Figure 17):

**Subclass 1:** *Unobservable* agent failure can be defined as a failure of the agent that cannot be proven to have occurred by all possible observers of inputs and outputs of the agent.

**Subclass 2:** *Observable* agent failure can be defined as a failure that an observer of inputs and outputs of the agent can prove has occurred.

Unobservable failures can have two possible reasons that are again unobservable:

- **Internal failure** - a failure in an internal processing of an agent that is not observable from outside the agent. The agent has been able to either recover from the failure or use an alternative way to avoid the manifestation of the failure.

- **Hidden fault** - a fault in the implementation of an agent that has not manifested yet. After the fault gets triggered, the fault transforms into another type of failure, possibly an internal failure.

The observable failures can be further classified as:

**Subclass 2.1:** *Behavior failure* - manifests when an agent does not behave as expected according to a given specification.

**Subclass 2.2:** *Knowledge failure* - appears when an agent deals with or provides incorrect, irrelevant, or old information according to a given specification for a particular situation.

The behavior failure manifests itself as one of the following types of failures:

**Subclass 2.1.1:** *No response* - an agent fails to respond to a particular request in a given timeframe for the response.

**Subclass 2.1.2:** *False response* - an agent responds to a particular request in a different way than noted in the given specification of the communication protocol.

**Subclass 2.1.3:** *Delayed response* - an agent responds to a particular request in a given timeframe for response but slower than noted in a given specification.

**Subclass 2.1.4:** *False request/inform* - an agent sends a request or an inform that is not correct according to the given specification.

**Subclass 2.1.5:** *Bogus requests/informs* - an agent sends numerous requests or informs that can be considered as fussing.
Subclass 2.1.6: **Proactive failure** - an agent either fails to take an action or chooses an incorrect action, according to a given specification, in a given situation, and in a current internal state.

Subclass 2.1.7: **Exterior failure** - an agent fails to behave correctly, according to a given specification, other than in communication among agents. For instance, an agent agreed upon some action and failed to finish this action in a given timeframe due to an external failure. Another example is that the agent agreed to change its location and failed to do it.

Furthermore, the no response failure, which is classified as an observable failure, can have several reasons that do not have to be necessarily observable:

- **Busy** - an agent is working on some task that has at least the same or higher priority than an incoming request (if there is such a priority), the internal processing of agent encountered an infinite processing loop that cannot be resolved, or a deadlock of resources occurred that cannot be resolved.

- **Major failure** - an agent crashed for some reason and is no longer available. This usually means that there is a fault in the implementation of the agent, but this fault can have its roots in an incorrect specification of the agent.

- **Communication failure** - an agent did not receive the request for some reason, the request has been corrupted by the transportation, or the request has been significantly delayed by the communication system resulting in an impossibility to respond in a given timeframe.

- **Hardware failure** - hardware that hosts the agent is not functioning correctly.

- **Not understood** - an agent did not understand the request (although all available knowledge to understand it has been used).

Knowledge failures can also be further classified using the classification of faults in knowledge bases [7]. We omit the intractability knowledge failure since it is not observable and, according to the presented classification of agent failures, it is either a reason why a knowledge failure occurred or a type of unobservable internal failure. Thus we classify knowledge failures as:

Subclass 2.2.1: **Incorrect knowledge** - some part of knowledge contains information that is not correct.

Subclass 2.2.2: **Incomplete knowledge** - some part of knowledge that should be present is missing. For instance, the message of type request-for-search does not contain the subject that should be searched for.

Subclass 2.2.3: **Inconsistent knowledge** - knowledge or a part of knowledge that is complete and correct per se, but in a broader view and/or in the current situation it is not correct.
Subclass 2.2.4: Redundant knowledge - knowledge or a part of knowledge that can be found or extracted elsewhere. The redundant knowledge can be, for instance, in a message that is used to register an agent and this message contains the same capabilities of the agent twice.

All four knowledge failures -- incorrect knowledge, incomplete knowledge, inconsistent knowledge, and redundant knowledge -- all classified as observable failures -- can have several reasons that do not have to be necessarily observable. The following list covers all five main ways to generate knowledge failures presented in [79].

- **Acquire failure** - an agent for some reason failed in the creation of knowledge.
- **Operational failure** - an agent for some reason failed in the execution of a knowledge base that resulted in incorrect knowledge.
- **Failed to find fault** - the internal inspection of the agent failed to find incorrect knowledge.
- **Failed to fix fault** - an agent has been unable to correct incorrect knowledge.
- **Failed to preserve** - during a knowledge fixing operation an agent corrupted knowledge that was previously correct.

### 7.2 Multi-Agent System Failure Types

All of the agent failure categories provide a framework to classify failures of a single agent. If we consider the whole multi-agent system, we cannot simply assume that failures of the multi-agent system consist just of failures of a single agent. The multi-agent system can be designed either by applying the connectionist model or using the concept of a hierarchical collective system [38].

The **connectionist model** is based on the fact that the creator of the system designs the behavior of components (agents) instead of designing the behavior of the whole system. As the components are connected together, they create an emergent behavior that is possibly much more complex than if we sum up the behavior of the components. As the system creator does not design the behavior of the whole system but only the behavior of system components, this approach can lead to mistakes that cannot be considered as a fault in any particular component (see section 7.1 for definitions of mistake, fault, etc.). In this case the fault that manifests in a failure can be located in the emergent behavior.

The concept of a **hierarchical collective system** is based on a top-down design of the system. The system creator designs a rough large task that is divided into smaller and finer subtasks. By using this concept, the system creator has a possibility to identify the appropriate fault when a failure occurs. In another words,
if there is no emergent behavior in the system, then fault in this behavior simply cannot exist.

Agents are not running in isolation, but are a part of a multi-agent system architecture. This architecture provides time and space to run the agents and a communication system, usually with given languages, protocols, and ontology definitions, that allows the agents to cooperate. These elements not only provide the possibility to create agents (components), but also represent the glue that connects them together.

Based on these remarks, we can now define possible types of multi-agent system failures.

- **Agent failure** - any failure of an agent implicitly means a failure within the multi-agent system (see section 7.1). This failure does not necessarily mean the failure of the whole multi-agent system. For instance, any unobservable failure of an agent cannot evolve into the failure of the whole system.

- **Agent environment failure** - the agent environment is not functioning properly. For instance, an agent tries to create a timer that is provided by the agent environment, but the creation failed for some reason.

- **Incorrect emergent behavior** - the agents do not behave as expected although the behavior of each particular agent is according to a specification for a given situation and given internal states. This usually means that the designer of the specification did not consider a given situation, internal states, or a combination of these. In another words, the agents encountered an unexpected situation and did not behave as expected.

- **Missing capability** - an agent that is running within a multi-agent system fails in the search for a capability provider. This failure occurs when the system is missing an agent with the desired capability (an agent failure is not considered) and agents are dependent on this capability.

- **Missing knowledge** - an agent is unable to obtain some particular knowledge using all available requests at its disposal. This failure occurs when the multi-agent system is not designed correctly to store or to provide this knowledge. Again, agent failure is not considered here.

- **Hardware failure** - any failure of hardware other than the hardware on which agents are running (since this is considered a type of agent failure), for instance, the failure of communication media.

- **Real-time response failure** - a multi-agent system fails to provide correct outputs in a given timeframe, occurring when the real time reaction of a multi-agent system on a particular set of inputs is slower or faster than expected.
- **Robustness failure** - a multi-agent system fails to provide the desired or expected fault-tolerant behavior after some other type of failure occurs.

Note that in all of these multi-agent system failure types, except for the agent failure and the robustness failure, no particular agent failed in the system at runtime.

### 7.3 Static Impact of Middle-Agent Failure

We define the following failure impact attributes to describe and assess the static effects of failures of middle-agents on loss of social knowledge about end-agents (see section 3.2.2):

**Definition 7.1**

- \( F_{i_k} \) - minimal k-failure impact - a minimal percentage of end-agents about which social knowledge stored in middle-agents is lost when \( k \) middle-agents fail simultaneously, where \( k \) is a natural number.

- \( F_{I_k} \) - maximal k-failure impact - a maximal percentage of end-agents about which social knowledge stored in middle-agents is lost when \( k \) middle-agents fail simultaneously, where \( k \) is a natural number.

Since the impact of the failure is usually dependent on many factors that can be hard to predict or to determine, we define minimal (optimistic) and maximal (pessimistic) values instead of just one.

One example of the failure impact attributes is as follows. Assume that a multi-agent system contains three middle-agents and one other middle-agent duplicating the entire knowledge of the first three. Middle-agent1 serves 20% of the end-agents, Middle-agent2 serves 30%, and the last one, Middle-agent3, serves the remaining 50% of agents. In this case, both \( F_{i_1} \) and \( F_{I_1} \) will be equal to 0% since when one failure occurs, the duplicating agent is still available and it covers the same social knowledge. \( F_{i_2} \) is also equal to 0% because the minimal impact of two end-agents failing is the case which both Middle-agent1 and Middle-agent2 fail simultaneously. \( F_{I_2} \) is equal to 50% for the worst case, when Middle-agent3 and the duplicating agent fail simultaneously.

To define the failure impact more precisely, graph theory [19] can be used to define equations that do not have restrictions on their usage.

**Definition 7.2** A graph \( G \) will be called \( M-E \) if \( G \) is a bipartite graph where partite sets are denoted as \( V_m(G) \) and \( V_e(G) \). Each middle-agent \( i \) is represented by a graph vertex \( v_i \in V_m(G) \) and end-agent \( j \) is represented by a graph vertex \( w_j \in V_e(G) \). If a middle-agent \( i \) holds social knowledge about an end-agent \( j \) then the graph \( G \) contains an edge \( e = \{v_i, w_j\} \). A set of edges of graph \( G \) is denoted as \( E(G) \) and a set of vertices as \( V(G) \). The number of middle-agents \( |V_m(G)| \) is denoted as \( M \) and the number of end-agents \( |V_e(G)| \) as \( N \).
Then we define following equations to compute minimal and maximal 1-failure impact:\(^\text{7}\):

\[
FI_i = \frac{100}{N} \cdot \min_{i=1}^{M} \left\{ \sum_{a=1}^{N} \text{COB}(a,i) \right\} \quad [\%] \tag{1}
\]

\[
FI_i = \frac{100}{N} \cdot \max_{i=1}^{M} \left\{ \sum_{a=1}^{N} \text{COB}(a,i) \right\} \quad [\%] \tag{2}
\]

Where \(\text{COB}(a,i)\) means that an end-agent \(a\) is ‘covered only by’ a middle-agent \(i\). This function is defined as follows:

\[
\text{COB}(a,i) = \begin{cases} 0, & \exists k \in \{v_k, w_a\} \in E(G) \land k \neq i \\ 1, & \text{otherwise.} \end{cases}
\]

An additional failure does not decrease the failure impact, which means that \(FI_N \leq FI_{N+1}\) and \(FI_N \leq FI_{N+1}\). Similarly, we define the following equations for two failures that occur simultaneously:

\[
FI_2 = \frac{100}{N} \cdot \min_{i,j=1}^{M} \left\{ \sum_{a=1}^{N} \text{COB}(a,i,j) \right\} \quad [\%] \tag{3}
\]

\[
FI_2 = \frac{100}{N} \cdot \max_{i,j=1}^{M} \left\{ \sum_{a=1}^{N} \text{COB}(a,i,j) \right\} \quad [\%] \tag{4}
\]

Where \(\text{COB}(a,i,j)\) means that an end-agent \(a\) is ‘covered only by’ middle-agents \(i\) and \(j\). This function is defined as follows:

\[
\text{COB}(a,i,j) = \begin{cases} 0, & \exists k \in \{v_k, w_a\} \in E(G) \land k \neq i \land k \neq j \\ 1, & \text{otherwise.} \end{cases}
\]

The same types of equations can be used to define the middle-agent failure impact for more than two failures.

The failure impact attributes can be used to rate how much information is the architecture able to protect in the case of the simultaneous failure of exactly \(M\) middle-agents. Nevertheless, to show the number of simultaneous failures of middle-agents needed to lose social knowledge about an end-agent, we define the following minimal redundancy.

**Definition 7.3** The minimal redundancy \(MR(G)\) of the graph \(G\) of type M-E is defined as follows:

\[
MR(G) = \min_{v \in V_e(G)} \left\{ d_G(w) \right\} \quad [5]
\]

\(^7\) Note that implicitly \(N > 0\) and \(M > 0\) since \(G\) is of type M-E and a bipartite graph has \(V_e(G) \neq \emptyset\) and \(V_m(G) \neq \emptyset\).
We would expect that MR has monotonic behavior (or is at least a constant function) over the six possible basic operations that change $G$.

- **Add an edge to $E(G)$** thus MR($G$) should not be decreased. Assume that $E(H) = E(G) \cup \{e\}$. Then either MR($H$) = MR($G$) or MR($H$) = MR($G$) + 1 holds, i.e., MR($H$) $\geq$ MR($G$). Thus MR is monotonic towards an edge addition and order-preserving.

- **Add a vertex to $V_e(G)$** thus MR($G$) should not be increased. Assume that $V(H) = V(G) \cup \{w\}$ where $w \in V_e(H)$. Then MR($H$) = 0, thus MR($H$) $\leq$ MR($G$) since MR($G$) $\geq$ 0. Thus MR($G$) is monotonic towards a vertex addition to $V_e(G)$ and order-reversing.

- **Add a vertex to $V_m(G)$** thus MR($G$) should not change. Assume that $V(H) = V(G) \cup \{v\}$ where $v \in V_m(H)$. Then MR($H$) = MR($G$). Thus MR($G$) is a constant function towards a vertex addition to $V_m(G)$.

- **Remove an edge from $E(G)$, remove a vertex from $V_e(G)$,** and **remove a vertex from $V_m(G)$** - can be proved in a similar fashion as the three cases above.

To provide a measure that can be used without specifying the number of simultaneous failures in which we are interested, we define average redundancy.

**Definition 7.4** The average redundancy AR($G$) of the graph $G$ of type M-E is defined as follows:

\[
AR(G) = \frac{1}{N} \sum_{v \in V_e(G)} d_G(v) = \frac{1}{N} \sum_{w \in V_e(G)} d_G(w) = \frac{1}{N}|E(G)|
\]  

(6)

We would expect that AR has monotonic behavior (or is at least a constant function) over the six possible basic operations that change $G$.

- **Add an edge to $E(G)$** thus AR($G$) should be increased. Assume that $E(H) = E(G) \cup \{e\}$. Then AR($H$) = AR($G$) + $1/N$, i.e., AR($H$) $>$ AR($G$). Thus AR is strictly increasing towards an edge addition.

- **Add a vertex to $V_e(G)$** thus AR($G$) should be decreased. Assume that $V(H) = V(G) \cup \{w\}$ where $w \in V_e(H)$. Then AR($H$) = AR($G$)$ \cdot N/(N+1)$, i.e., AR($H$) $<$ AR($G$). Thus AR($G$) is strictly decreasing towards a vertex addition to $V_e(G)$.

---

8 Measure is used only as a general term in this context. In mathematical measure theory, a measure is a function which assigns to every element of a given $\sigma$-algebra a non-negative real number or $\infty$, where the empty set has measure zero and the measure is countably additive. A family of bipartite graphs is not suitable to be defined as a $\sigma$-algebra since a union of bipartite graphs has to be defined in this case in such a way that these two graphs must have common vertices; that is a very significant restriction.
• Add a vertex to $V_m(G)$ thus $AR(G)$ should not change. Assume that $V(H) = V(G) \cup \{v\}$ where $v \in V_m(H)$. Then $AR(H) = AR(G)$. Thus $AR(G)$ is a constant function towards a vertex addition to $V_m(G)$.

• Remove an edge from $E(G)$, remove a vertex from $V_e(G)$, and remove a vertex from $V_m(G)$ - can be proved in a similar fashion as the three cases above.

If there is information available about the importance of end-agents, it can be associated with the vertices or edges of the graph. Therefore we also define weighted average redundancy as follows.

**Definition 7.5** Assume that $W(w)$ is a function that assigns the weight of a vertex $w \in V_e(G)$, where $W(w) \in (0, \infty)$. Then the weighted average redundancy $AR_W(G)$ of the graph $G$ of type M-E is defined as follows:

$$AR_W(G) = \frac{1}{N} \sum_{w \in V_e(G)} W(w) \cdot d_G(w)$$

We would expect that $AR_W$ has monotonic behavior (or is at least a constant function) over the six possible basic operations that change $G$.

• Add an edge to $E(G)$ thus $AR_W(G)$ should be increased. Assume that $E(H) = E(G) \cup \{e\}$, where $e = \{v, w\}$. Then $AR_W(H) = AR_W(G) + W(w)/N$, i.e., $AR_W(H) > AR_W(G)$ since $W(w) > 0$. Thus $AR_W$ is strictly increasing towards an edge addition.

• Add a vertex to $V_e(G)$ thus $AR_W(G)$ should be decreased. Assume that $V(H) = V(G) \cup \{w\}$ where $w \in V_e(G)$. Then $AR_W(H) = AR_W(G) \cdot N/(N+1)$, i.e., $AR_W(H) < AR_W(G)$. Thus $AR_W(G)$ is strictly decreasing towards a vertex addition to $V_e(G)$.

• Add a vertex to $V_m(G)$ thus $AR_W(G)$ should not change. Assume that $V(H) = V(G) \cup \{v\}$ where $v \in V_m(G)$. Then $AR_W(H) = AR_W(G)$. Thus $AR_W(G)$ is a constant function towards a vertex addition to $V_m(G)$.

• Remove an edge from $E(G)$, remove a vertex from $V_e(G)$, and remove a vertex from $V_m(G)$ - can be again proved in a similar fashion as the three cases above.

We will use these definitions of middle-agent failure impact and redundancy to assess static robustness of various knowledge propagation methods (see section 8.1) and in the experimental section (see section 9.2) to provide a comparison of the robustness of different types of architectures.

### 7.4 Dynamic Impact of Middle-Agent Failure

In the previous section we described how much information is lost in the case of simultaneous failures of middle-agents, which is a static effect. There is
also a dynamic effect that is associated with the failures. Assume that in a structure of middle-agents that are interconnected by communication channels a failure of several middle-agents occurs. As a result, further information propagation through this damaged structure can be reduced or even impossible.

The structure of the middle-agents can be described by graph theory and graph attributes already exist that describe the fault tolerance of a graph.

A graph $G$ is said to be $\lambda$ vertex-connected (or $\lambda$-connected), for $\lambda \in \mathbb{N}$, if $|V(G)| > \lambda$ and the graph after removal of a set of vertices $X$ from $V(G)$ is connected for every set $X \subseteq V(G)$ with $|X| < \lambda$ [19], i.e., if the deletion of at most $\lambda - 1$ vertices leaves the graph connected, or if there are $\lambda$ many vertex disjoint paths between any two arbitrary vertices. The greatest integer $\lambda$ such that $G$ is $\lambda$ vertex-connected is the connectivity $k(G)$ of $G$.

A graph is called $\lambda$ edge-connected if $|V(G)| > 1$ and a graph after removal of a set of edges $F$ from $E(G)$ is connected for every set $F \subseteq E(G)$ of fewer than $\lambda$ edges [19], i.e., if the deletion of at most $\lambda - 1$ edges leaves the graph connected. The greatest integer $\lambda$ such that $G$ is $\lambda$ edge-connected is the edge-connectivity $\lambda(G)$ of $G$.

Therefore, the structure of middle-agents that is described by graph theory can be analyzed for its vertex and edge connectivity to prove how many failures of either middle-agents or communication channels can occur simultaneously without losing the possibility to fully route messages through the structure.
8 New Structure of Social Knowledge Distribution

In section 5 we summarized existing centralized, distributed, and hybrid types of social knowledge distribution in a multi-agent system. Only the hybrid type satisfies the requirement to design a structure of social knowledge distribution that is more robust than the centralized type and more scalable than the distributed type.

Both the facilitator approach and the teamwork-based techniques are based on a grouping of social knowledge into a fully interconnected structure of middle-agents, where the number of middle-agents is significantly less than the number of end-agents. This full interconnection is a significant drawback of these architectures.

The distributed matchmaking approach uses several types of middle-agents that are distributed, but there is one root matchmaking agent which is a single point of failure and also a communication bottleneck. The same holds for various distributed blackboards architectures that have several specialized interconnected blackboards with one global blackboard.

In the new design we start from the idea of a hierarchical approach. The scalability of the hierarchical approach is increased in comparison with the distributed approaches. The distributed approaches usually need \( N(N-1)/2 \) interconnections, where \( N \) is the number of middle-agents in the system. The hierarchical approaches usually need only \( N-1 \) interconnections. The price paid, however, is the decrease in system robustness since the connection among vertices (middle-agents in our case) is not redundant. If any non-local middle-agent fails, then the whole structure is affected by this failure to some extent.

The hierarchical structure can also be described by the terminology of graph theory [19] as a *tree*, with the middle-agents as the tree vertices and a graph edge corresponding to communication between two middle-agents. A tree is minimally connected, i.e., by removing any edge from the tree the graph becomes disconnected. For a system to be robust, it must be more than minimally connected.

In the hierarchical approach we start from a distributed system and group several end-agents around one local middle-agent. The grouping is usually performed according to a physical or functional (task-oriented) distribution of agents.

One example of physical distribution is that all agents that reside within one autonomous execution unit (e.g., programmable controller) are assigned to the one middle-agent that also resides in this execution unit. Another example is that all agents located on one local area network are grouped together.
An example of a functional distribution is that agents that deal with customer orders are located in one group, agents focused on repository management are located in the second group, and so on.

8.1 Dynamic Hierarchical Teams

We propose the dynamic hierarchical teams (DHT) architecture [130] to take advantage of both hierarchical and distributed architectures. The pure hierarchical architectures offer the scalability, but they are not designed to be fault-tolerant. On the other hand, the pure distributed architectures offer the robustness, but they are not scalable since any middle-agent is connected to all other middle-agents.

8.1.1 Dynamic Hierarchical Teams Architecture Description

Assume that a multi-agent system consists of middle-agents and end-agents (see section 3.2.2). The middle-agents can form a structure that can be described by graph theory. Graph vertices represent middle-agents and graph edges represent communication channels between two middle-agents.

The first main difference from the pure hierarchical architecture is that the DHT architecture is not restricted to have a single root of the tree that serves as a global middle-agent. The single global middle-agent easily becomes a single point of failure and possibly also a communication bottleneck. In addition, any other middle-agent that is not in a leaf position in the tree has similar disadvantages.

For instance, assume that the structure of middle-agents consists of one global middle-agent as a root of the tree and at least two other middle-agents that are connected to this root. In this case the global middle-agent manages communication between the child middle-agents, because the architecture does not allow direct communication among the child middle-agents. A failure in the global middle-agent can cause a split of the whole structure into several subsections, each without the possibility to further communicate to the others. Similarly, if the tree is larger then any other failed middle-agent that is not in a leaf position in the graph can cause the split of its child middle-agents.

Therefore, to provide a more robust architecture, each middle-agent that is not a leaf in the tree should be backed up by another middle-agent. This is similar to the warm and hot backup technique (see section 6.3.1.1). The difference is that in this case the secondary middle-agent is not observing the primary one for a failure, but they are both fully functional parts of the whole system. Groups of these middle-agents we call *teams* (see Figure 18). Whenever one of the middle-agents from the team fails, other middle-agents from the team can subrogate this agent.

9 This behavior can be simply characterized by the well-known sentence ‘All for one and one for all’ - Dumas Alexandre: *The Three Musketeers*. Penguin Books, New York, 1986.
During the normal operation of the DHT structure all middle-agents use only the primary communication channels. The use of the secondary communication channels will be further described in section 8.1.5.

The DHT structure is not limited only to two levels, but it can support an $N$-level structure. For the cases where $N > 2$ the teams (not the middle-agents) compose the hierarchical structure in the form of a tree (see Figure 19), i.e., the structure of teams does not contain circuits and the graph is connected. The tree structure holds only if we consider only one edge of the resulting graph per a set of primary and secondary connections between two teams.

**Figure 19: Example of 3-level DHT architecture and associated structure of teams**

Much more complicated is the structure of the middle-agents (not the teams), since this structure is not limited to be a tree. First of all, a team consists of at least one middle-agent. To increase fault tolerance, a team should consist of two or more middle-agents. All members of a team are interconnected via communication channels, forming a complete graph (see graph theory [19]). The members of the topmost team (Team1) are interconnected via primary communication channels while the members of other teams are interconnected via secondary ones.
If we restrict the DHT structure to contain only teams that consist of only one middle-agent then we end up with a hierarchical structure (a tree). On the other hand, if we restrict it to one team plus possibly one middle-agent that is not a part of this team then a full-connected network of middle-agents is created, i.e., a structure similar to the teamwork-based technique (see 5.3.2). The DHT structure is therefore flexible in this respect.

Let $G$ be a graph where each middle-agent $i$ in the DHT structure is represented by a graph vertex $v_i \in V$ and each primary or secondary connection among middle-agents $i$ and $j$ is represented by an edge $e = \{v_i, v_j\}$ between $v_i$ and $v_j$.

**Definition 8.1** A graph $G$ will be called a **DHT** graph if there exist non-empty sets $V_1, \ldots, V_n \subset V(G)$ such that they are pairwise disjoint and $V_1 \cup \ldots \cup V_n \neq V(G)$. In that case, the complete subgraph $G_i$ of the graph $G$ induced by the set of vertices $V_i$ will be called a **team** of $G$ if all of the following is satisfied:

1) $\forall v \in V(G) \setminus V_1 \rightarrow \exists w (w \in V_j \rightarrow \{v, w\} \in E(G))$

2) $\forall v \in V(G) \land v \notin V_1 \cup V_2 \cup \ldots \cup V_n \rightarrow \exists j \forall w (w \notin V_j \rightarrow \{v, w\} \notin E(G))$

3) $\forall j((j > 1) \land (j \leq n) \rightarrow \exists k((k < j) \land \forall v \forall w (v \in V_j \land w \in V_k \rightarrow \{v, w\} \in E(G))) \land \forall u \forall m (u \in V_m \land (m < j) \land (m \neq k) \rightarrow \{v, u\} \notin E(G)))$

**Definition 8.2** The graph $G$ is called **DHT-$\lambda$** if $G$ is DHT and $|V_i| = \lambda$ for every $i = 1, \ldots, n$, where $\lambda \in N$.

### 8.1.2 Fault Tolerance in Dynamic Hierarchical Teams

The fault tolerance of an undirected graph is measured by the vertex and edge connectivity of the graph. We already described that the vertex and edge connectivity of a graph can be used to determine the dynamic impact of failure of middle-agents (see section 7.4).

To briefly summarize these terms, a graph $G$ is said to be $\lambda$ vertex-connected if the deletion of at most $\lambda - 1$ vertices leaves the graph connected. The greatest integer $\lambda$ such that $G$ is $\lambda$ vertex-connected is the connectivity $\kappa(G)$ of $G$. A graph is called $\lambda$ edge-connected if the deletion of at most $\lambda - 1$ edges leaves the graph connected. The greatest integer $\lambda$ such that $G$ is $\lambda$ edge-connected is the edge-connectivity $\lambda(G)$ of $G$.

---

10 For all vertices $v$ of $G$ except $V_1$ there has to be a team such that $v$ is connected to all members of this team.

11 For all vertices $v$ that are not members of any team there are only connections to one team and there cannot be any other connection from $v$.

12 All members of each team except $G_i$ are connected to all members of exactly one other team with lower index.
Claim 8.1 \textit{If the graph }G\textit{ is DHT-}\lambda\textit{ then for each vertex }v \in V(G)\textit{ there exists a vertex }w \in V_1\textit{ such that there is a path in }G\textit{ starting from }v\textit{ and ending at }w\textit{ after removing }\lambda - 1\textit{ vertices or }\lambda - 1\textit{ edges from }G.

\textit{Proof.} Assume the case where }v \notin V_1\textit{ since otherwise the path is }v\textit{ itself. For each team of DHT-}\lambda\textit{ }|V_j| = \lambda. \textit{Thus there are at least }\lambda\textit{ edges }\{v, w_1\}\textit{ such that }\exists j \forall v_1 (v_1 \in V_j \rightarrow \{v, w_1\} \in E(G)). \textit{Since }|V_j| = \lambda\textit{ then after the elimination of }\lambda - 1\textit{ vertices or }\lambda - 1\textit{ edges there exists a path starting from }v\textit{ and ending at }w_1\textit{ where }w_1 \in V_j. \textit{If }j = 1\textit{ then the resulting path is }v w_1. \textit{Otherwise, since }|V_j| = \lambda\textit{ the rule 3) from the definition of DHT can be repeatedly applied to construct a path in }G\textit{ despite the elimination of }\lambda - 1\textit{ vertices or }\lambda - 1\textit{ edges starting from }w_1\textit{ and ending at }w_k\textit{ where }w_k \in V_1. \textit{Therefore the resulting path in this case is }v w_1 w_2 \ldots w_k. \square

Lemma 8.1 \textit{If the graph }G\textit{ is DHT-}\lambda\textit{ then }G\textit{ is }\lambda\textit{ vertex-connected and }\lambda\textit{ edge-connected.}

\textit{Proof.} 1) \textit{We prove that the graph }G\textit{ of type DHT-}\lambda\textit{ is }\lambda\textit{ edge-connected. Suppose (for contradiction) that there is a }\lambda - 1\textit{ edge cut set in }G. \textit{Assume that it separates }G\textit{ into pieces }C_1\textit{ and }C_2. \textit{Let }v_1 \in V(C_1)\text{ and }v_2 \in V(C_2). \textit{We already proved that after removing }\lambda - 1\textit{ edges there exists a path starting from a vertex }v_1\textit{ (or }v_2\textit{ respectively) and ending at }w_1\textit{ (or }w_2\textit{ respectively) where }w_1 \in V_1\text{ and }w_2 \in V_1. \textit{If }w_1 = w_2\textit{ then a path from }v_1\text{ to }v_2\textit{ already exists. Otherwise it remains to prove that any two vertices }w_1 \neq w_2\textit{ such that }w_1 \in V_1\text{ and }w_2 \in V_1\textit{ are connected after elimination of }\lambda - 1\textit{ edges from }G. \textit{At least }\lambda - 1\textit{ edge cut set is required to split the complete graph }G_1\textit{ into two pieces but since }G_1 \neq G\textit{ thus }\exists w_3 (w_3 \notin V_1 \land w_3 \in V(G))\textit{ for which }\forall v_2 (v_2 \in V_1 \rightarrow \{w_3, v_2\} \in E(G))\textit{ holds since either }w_3 \in V_2\text{ or the number of teams }n = 1. \textit{Then a subgraph of }G\textit{ induced by }V(G_1) \cup \{w_3\}\textit{ is a complete graph of order }\lambda + 1\textit{ and therefore there is at least one path in }G\textit{ after elimination of }\lambda - 1\textit{ edges from }G\textit{ that leads from }w_1\text{ to }w_2. \textit{Thus there is no edge cut set of size }\lambda - 1. \square

2) \textit{We prove that the graph }G\textit{ of type DHT-}\lambda\textit{ is }\lambda\textit{ vertex-connected. Suppose (for contradiction) that there is a }\lambda - 1\textit{ vertex cut set in }G. \textit{Assume that it separates the graph at least into pieces }C_1\textit{ and }C_2. \textit{Let }v_1 \in V(C_1)\text{ and }v_2 \in V(C_2). \textit{We already proved that after removing }\lambda - 1\textit{ vertices there exists a path starting from a vertex }v_1\textit{ (or }v_2\textit{ respectively) and ending at }w_1\textit{ (or }w_2\textit{ respectively) where }w_1 \in V_1\text{ and }w_2 \in V_1. \textit{If }w_1 = w_2\textit{ then a path from }v_1\text{ to }v_2\textit{ already exists. Otherwise since }G_1\textit{ is a complete graph then any two vertices }w_1 \neq w_2\textit{ where }w_1 \in V(G_1)\text{ and }w_2 \in V(G_1)\textit{ are connected after elimination of }\lambda - 1\textit{ vertices from }G. \textit{Therefore there is no vertex cut set of size }\lambda - 1. \square

Claim 8.2 \textit{If the graph }G\textit{ is DHT-}\lambda\textit{ then the minimum degree }\delta(G) = \lambda.

\textit{Proof.} We already proved that }G\textit{ is }\lambda\textit{ edge-connected and therefore }\delta(G) \geq \lambda. \textit{From the definition of DHT rule 1) (see Definition 8.1) and the fact that }V_1 \cup \ldots \cup V_n \neq V(G)\textit{ there has to be at least one vertex }v' \in V(G)\textit{ for which }v' \notin V_1 \cup \ldots \cup V_n\textit{ holds and }\exists \forall w (w \in V_j \rightarrow \{v', w\} \in E(G)). \textit{Since }|V_j| = \lambda\textit{ for DHT-}\lambda\textit{ thus there are at least }\lambda\textit{ edges }\{v', w\}\textit{ and since from the definition of DHT rule 2) }\exists ! j \forall w (w \notin V_j \rightarrow
{v', w} \notin E(G) \) holds there are no more than \( \lambda \) edges \{v', w\} and thus \( d(v') = \lambda \). Since \( \delta(G) \geq \lambda \) and \( d(v') = \lambda \) therefore \( \delta(G) = \lambda \).

**Theorem 8.1** If the graph \( G \) is DHT-\( \lambda \) then the **vertex connectivity** \( \kappa(G) = \lambda \) and the **edge-connectivity** \( \lambda(G) = \lambda \).

**Proof.** We already proved that the graph \( G \) of type DHT-\( \lambda \) is \( \lambda \) vertex-connected and \( \lambda \) edge-connected (see Lemma 8.1) therefore it remains to prove that \( \kappa(G) \leq \lambda \) and \( \lambda(G) \leq \lambda \). For every non-trivial graph \( G \) the equation \( \kappa(G) \leq \lambda(G) \leq \delta(G) \) holds [19]. We already proved that \( \delta(G) = \lambda \) (see Claim 8.2) thus \( \kappa(G) \leq \lambda \) and \( \lambda(G) \leq \lambda \).

The DHT structure where teams consist of \( \lambda \) middle-agents is therefore fault tolerant to simultaneous failure of at least \( \lambda - 1 \) middle-agents and also to simultaneous failure of at least \( \lambda - 1 \) communication channels.

A graph \( G \) is called **maximally fault tolerant** if vertex connectivity of the graph \( G \) equals the minimum degree of a graph \( \delta(G) \) [132].

**Theorem 8.2** The graph \( G \) of type DHT-\( \lambda \) is maximally fault tolerant.

**Proof.** We already proved that the graph \( G \) of type DHT-\( \lambda \) has vertex connectivity \( \kappa(G) = \lambda \) (see Theorem 8.1) and we also proved that it has the minimum degree \( \delta(G) = \lambda \) (see Claim 8.2).

The maximally fault tolerant graph means that there is no bottleneck in the structure of connections among vertices, i.e., middle-agents in the case of the DHT architecture.

Moreover, \( N \)-version programming (see section 6.3.1.2) can be directly applied to the DHT architecture to increase robustness of the whole system. Assume that the teams are of size \( N \) and consist of \( N \) independently developed middle-agents (by \( N \) different developers, by \( N \) different programming languages, etc.). Then the whole structure of middle-agents is fault tolerant to the simultaneous failure of all middle-agents that were developed by \( N-1 \) development processes.

### 8.1.3 Social Knowledge Management in Dynamic Hierarchical Teams

We identified several approaches to social knowledge management based on the amount of social knowledge that is stored in the low-level middle-agents in a hierarchy. In this section we describe breadth knowledge propagation, depth knowledge propagation, and no knowledge propagation. The efficiency of these three methods for social knowledge propagation can be further improved by knowledge propagation on demand or by knowledge caching.

To formally describe these knowledge propagation methods, we first define neighbors, parents, and team members of a middle-agent.
Definition 8.3 Assume that a graph $G$ is DHT with $G_1, \ldots, G_n$ its teams. Then we define all of the following:

1) $E^p(G) \subseteq E(G)$ is a set of edges where each $e \in E^p(G)$ represents a primary communication channel.

2) $\text{Neighbors}(v, G) = \{ w \mid \{v, w\} \in E^p(G) \}$.

3) $\text{Parents}(v, G) = \{ w \mid \{v, w\} \in E^p(G) \land \exists j \forall k(w \in V(G_j) \land v \in V(G_k) \rightarrow j < k).\}$

4) if $v \in V(G_j)$ then $\text{TeamMembers}(v, G) = V(G_j) \setminus \{v\}$
   if $v \notin V(G_j)$ for every $j = 1, \ldots, n$ then $\text{TeamMembers}(v, G) = \emptyset$.

8.1.3.1 Breadth Knowledge Propagation

We define breadth knowledge propagation in such a way that every middle-agent in the system ultimately knows social information about all end-agents in the system.

The following message routing algorithm called full message routing is used for routing messages in the breadth knowledge propagation approach and this algorithm holds for each middle-agent. This algorithm contains a number of subroutines that are only briefly defined since exact definition is implementation specific.

Definition 8.4 Assume that a graph $G$ is DHT. Let $m$ be a message instance that the middle-agent represented by vertex $v \in V(G)$ received from a middle-agent represented by vertex $v_{\text{orig}} \in V(G)$ or from an end-agent for which $v_{\text{orig}} \notin V(G)$. Let $\text{AddOrig}(m, V'(G))$ be a subroutine that stores a set of vertices $V'(G) \subset V(G)$ in the message $m$ and returns this result as a message $m'$. Let $V_{\text{orig}}(m) \subset V(G)$ be a set of vertices stored in the message $m$ such that $v \in V_{\text{orig}}(\text{AddOrig}(m, \{v\}))$.

Let $\text{Send}(H, m)$ be a subroutine that sends a message $m$ to all middle-agents that are represented by vertices $v \in H$ where $H \subset V(G)$. Let $\text{KB}(v)$ be an internal knowledge base of the middle-agent represented by a vertex $v$. Let $\text{Update}(v, m)$ be a subroutine that updates $\text{KB}(v)$ of the middle-agent $v$ based on message $m$. Let $\text{Store}(v, w, c)$ be a subroutine that stores a reference to the middle-agent $w$ under the message

---

13 A set of vertices where each ‘parent’ $w$ is connected to $v$ by primary communication channel and where $w$ belongs to the team with lower index than the index of the team where $v$ belongs to (if there is such team of $v$).

14 AddOrig subroutine is typically used to store a set of vertices into the message $m$ and then the resulting message $m'$ is sent to other middle-agents. The receiver of this message $m'$ can retrieve this set of vertices by $V_{\text{orig}}(m')$ and avoid to contact these middle-agents thus avoiding circuits in communication.

15 Update subroutine is one of the main parts of a middle-agent where information about end-agents, e.g., registration of capabilities of an end-agent, is processed and possibly stored to the internal knowledge base.
context $c$ into $KB(v)$ and let $Retrieve(v, c)$ be a subroutine that returns $H \subseteq V(G)$ where a vertex $w \in H$ iff $KB(v)$ contains $w$ under the message context $c$. Let $Context(m)$ be a subroutine that returns context of a message $m$, i.e., the same value for all messages that are successors of the original request message. Then the full message routing in $G$ is defined by the following algorithm that is performed by a middle-agent $v$ upon receiving a message $m$:

```plaintext
IF $Retrieve(v, Context(m)) \neq \emptyset$ THEN
    Update($v, m$)
    Let $R(v) = \{w \mid w \in Neighbors(v, G) \land w \neq v^{orig} \land w \notin V^{orig}(m)\}$
        be a set of potential receivers.
    Let $S(v) = \{w \mid w \in R(v) \land w \in TeamMembers(v, G)\}$
    FOR EACH $w \in R(v)$
    {
        IF $w \in TeamMembers(v, G)$ THEN $Send(\{w\}, m)$
        ELSE $Send(\{w\}, AddOrig(m, \{v\} \cup (S(v) \setminus \{w\}))$)
    }
    IF $R(v) \neq \emptyset$ THEN $Store(v, v^{orig}, Context(m))$.

Based on this message routing in the breadth knowledge propagation we can distinguish how different types of messages are propagated.

1. Registration, unregistration or modification messages are routed to all middle-agents via the full message routing.

2. A search request is replied to the sender by using only the locally stored knowledge.

When an end-agent anywhere in the system contacts a local middle-agent and passes registration information to it, this middle-agent updates its internal knowledge base based on the incoming message and propagates this information to all neighbor middle-agents over primary communication channels except the sender and except any middle-agent that is already mentioned in the incoming message to avoid loops of size less than four (see Figure 20). Since some of the communication channels can be faulty, the topmost team that consists of more than three middle-agents can have a loop of size greater or equal to four. Therefore the context of the message is used to avoid this type of loop. The breadth knowledge propagation approach holds for requests for registration or unregistration of an end-agent and also for the modification of social knowledge. Search requests can be handled by middle-agents locally since social knowledge about all end-agents in the system is ultimately present in every middle-agent.
Figure 20: Breadth knowledge propagation example

Figure 20 contains only the interactions that are directly related to the registration and to the search process.

1. E-Agent1 sends a registration request to its local middle-agent (M-A 4).
2. M-A 4 forwards the registration information to M-A 22.
3. M-A 22 forwards the registration information to M-A 12.
4. M-A 12 forwards the registration information simultaneously to M-A 11, 13, 21, and 23.
5. M-A 21, 23, and 13 forward the registration information simultaneously to M-A 5, 6, 31, 32, 33. Note that the primary connection between M-A 11 and 13 is not used at this time since it would close a circuit.
6. M-A 32 and 33 finally forward the registration information simultaneously to M-A 7, 8, and 9.
7. E-Agent2 sends a request to its local middle-agent (M-A 7) for a search.
8. M-A 7 can reply with the search result by using only locally stored knowledge.

Since by using the breadth knowledge propagation every middle-agent in the system ultimately knows social knowledge about all end-agents, the static impact of a middle-agent failure is as low as possible. Assume that vertices of the graph $G$ of type M-E (see section 7.3) represent all middle-agents and end-agents and an edge $\{v_i, w_j\} \in E(G)$ only for all middle-agents $i$ and end-agents $j$, i.e., the complete bipartite graph. Then the average redundancy and minimal redundancy (see section 7.3) are computed as follows:

$$\text{AR}(G) = \text{MR}(G) = \frac{1}{N} \sum_{v \in V_e(G)} d_G(v) = \frac{1}{N} \sum_{v \in V_e(G)} N = \sum_{v \in V_e(G)} 1 = M$$

Since the breadth knowledge propagation uses the full message routing, social knowledge about an end-agent is propagated using $N$ messages (if we do not
count possible acknowledgement messages), where $N$ is the number of middle-agents. Social knowledge is searched using 2 messages.

### 8.1.3.2 Depth Knowledge Propagation

The second approach to social knowledge management is depth knowledge propagation, in which a middle-agent propagates social knowledge only to the higher level of the hierarchy of teams. In this approach only the topmost middle-agents contain social knowledge about all end-agents in the system.

The following message routing algorithms are used for routing messages in the depth knowledge propagation approach and these algorithms hold for each middle-agent:

**Definition 8.5** Apply the same set of assumptions as for the full message routing. Then the root message routing in $G$ is defined by the following algorithm that is performed by a middle-agent $v$ upon receiving a message $m$ from $v^{\text{orig}}$:

$$\text{IF } \text{Retrieve}(v, \text{Context}(m)) \neq \emptyset \text{ THEN}$$

$$\begin{align*}
\text{Update}(v, m) \\
&\text{Let } R(v) = \{w \mid (w \in \text{Parents}(v, G) \cup \text{TeamMemembers}(v, G)) \land \} \\
&\{v, w\} \in E(G) \land w \neq v^{\text{orig}} \land w \notin V^{\text{orig}}(m)\} \\
&\text{Let } S(v) = \{w \mid w \in R(v) \land w \in \text{TeamMemembers}(v, G)\} \\
&\text{FOR EACH } w \in R(v) \\
&\begin{align*}
&\text{IF } w \notin \text{TeamMemembers}(v, G) \text{ THEN Send}(\{w\}, m) \\
&\text{ELSE Send}(\{w\}, \text{AddOrig}(m, \{v\} \cup (S(v) \setminus \{w\})))
\end{align*} \\
&\text{IF } R(v) \neq \emptyset \text{ THEN Store}(v, v^{\text{orig}}, \text{Context}(m))
\end{align*}$$

**Definition 8.6** Apply the same set of assumptions as for the full message routing. Also let $\text{Process}(v, m)$ be a subroutine that changes message $m$ based on information of middle-agent represented by vertex $v$ and returns true if all objectives of $m$ have been satisfied and false otherwise. Then the parent retrieval message routing in $G$ is defined by the following algorithm that is performed by a middle-agent $v$ upon receiving a message $m$ from $v^{\text{orig}}$:

$$\text{IF } \text{Retrieve}(v, \text{Context}(m)) \neq \emptyset \text{ THEN}$$

$$\begin{align*}
&\text{IF } \text{Process}(v, m) \text{ THEN Send}(\text{Retrieve}(v, \text{Context}(m)), m) \\
&\text{ELSE} \\
&\begin{align*}
&\text{FOR EACH } w \in \text{Parents}(v, G) \\
&\text{Send}(\{w\}, m)
\end{align*}
\end{align*}$$
**Social Knowledge in Multi-Agent Systems - New Structure of Social Knowledge Distribution**

```plaintext
IF Parents(v, G) ≠ ∅ THEN Store(v, v^{orig}, Context(m))
ELSE Send({v^{orig}}, m)
}
ELSE Send( Retrieve(v, Context(m)), m)
```

Based on this message routing in the depth knowledge propagation we can distinguish how different types of messages are propagated.

1. Registration, unregistration or modification messages are routed via the root message routing.
2. If a search request can be satisfied using local knowledge then reply with the result to the requester.
3. If a search request cannot be satisfied using local knowledge then it is routed via the parent retrieval message routing (if the set of receivers is non empty); otherwise, reply to the requester that the search was unsuccessful.
4. Forward the result of the search request back to the original requester (stored in v^{orig} under the same context as the result).

![Figure 21: Depth knowledge propagation example](image)

Figure 21 contains interactions that are directly related to the registration process and to the search process.

1. E-Agent1 sends a registration request to its local middle-agent (M-A 4).
2. M-A 4 forwards the registration information to M-A 22.
3. M-A 22 forwards the registration information to M-A 12.
5. E-Agent2 sends a search request to its local middle-agent (M-A 7).
6. M-A 7 forwards the search request to M-A 32 since the search request cannot be satisfied locally.
8. M-A 13 is able to satisfy the search request and replies with the search result back to M-A 32.
10. M-A 7 finally replies with the search result to E-Agent2.

The same mechanism is also used for social knowledge update, i.e., when an agent wants to add a new capability, remove a lost capability, change a capability description, or unregister from the community of agents.

The static impact of middle-agent failure in the dynamic hierarchical teamwork that uses the depth knowledge propagation is computed as follows. Assume that the graph $G$ is DHT. Assume that the vertices of the M-E graph $H = (V(H), E(H))$, where $V(H) \supset V(G)$, represent all end-agents and middle-agents and an edge $\{v_i, w_j\} \in E(H)$ if and only if middle-agent $i$ holds social knowledge about end-agent $j$. Assume that it is possible to compute the average number of vertices that have to be traversed by the root message routing to get from $v$ to $w$ (excluding $w$ and including $v$ if $v \neq w$) denoted as $\text{AvHeight}(G)$, where $v \in V(G)$ and $w \in V_1$ hold (see section 8.1.1 for definition of $V_1$). Assume that it is possible to compute the minimal number of vertices that have to be traversed by the root message routing to get from $v$ to $w$ denoted as $\text{MinHeight}(G)$. Then the average redundancy and minimal redundancy are computed as follows:

$$\text{AR}(H) = \frac{1}{N} \sum_{v \in V(H)} d_H(v) = \text{AvHeight}(G) + |V_1|$$

$$\text{MR}(H) = \min_{v \in V(H)} \{d_H(v)\} = \text{MinHeight}(G) + |V_1|$$

Since the depth knowledge propagation uses the root message routing, social knowledge about an end-agent is on average propagated using $\text{AvHeight}(G) + |V_1|$ messages (if we do not count possible acknowledgement messages). Social knowledge is searched via the parent retrieval message routing; therefore, it uses on average $2 \times \text{AvHeight}(G)$ messages.

### 8.1.3.3 No Knowledge Propagation

The last approach to social knowledge propagation is no knowledge propagation. In this approach the end-agents register, unregister, and modify registration information only at the local middle-agents; this information is not further propagated. This type of technique is used, for instance, in multi-agent systems that are FIPA compliant (see section 4.1.2) as JADE (see section 4.2.1).

**Definition 8.7** Apply the same set of assumptions as for the parent retrieval message routing. Let $\text{StoreExpectedReply}(v, w, c, m)$ be a subroutine that stores a reference to middle-agent represented by a vertex $w$ under the message context $c$ into $\text{KB}(v)$.
and $\text{RemoveExpectedReply}(v, w, c)$ be a subroutine that removes a reference to middle-agent represented by a vertex $w$ under the message context $c$ from $\text{KB}(v)$. Let $\text{RetrieveExpectedReplies}(v, c)$ be a subroutine that returns a set of vertices stored in $\text{KB}(v)$ under the message context $c$. Let $\text{AddReply}(v, c, m)$ be a subroutine that stores information from $m$ to the database of $v$ under the context $c$ and $\text{GetReply}(v, c)$ retrieves composite message based on previously stored information in $\text{KB}(v)$ under the message context $c$. Then the full retrieval message routing in $G$ is defined by the following algorithm that is performed by a middle-agent $v$ upon receiving a message $m$ from $v^{\text{orig}}$:

$$\text{IF Retrieve}(v, \text{Context}(m)) \neq \emptyset \text{ THEN}$$

$$\text{IF Process}(v, m) \text{ THEN Send(Retrieve}(v, \text{Context}(m)), m)$$

$$\text{ELSE}$$

$$\text{IF Process}(v, m) \text{ THEN Send(Retrieve}(v, \text{Context}(m)), m)$$

$$\text{ELSE}$$

$$\begin{align*}
\text{Let } R(v) &= \{w \mid w \in \text{Neighbors}(v, G) \land w \neq v^{\text{orig}} \land w \neq v^{\text{orig}}(m)\} \\
\text{Let } S(v) &= \{w \mid w \in R(v) \land w \in \text{TeamMembers}(v, G)\} \\
\text{FOR EACH } w \in R(v) \\
\{ \\
\text{StoreExpectedReply}(v, w, \text{Context}(m)) \\
\text{IF } w \in \text{TeamMembers}(v, G) \text{ THEN Send(}\{w\}, m) \\
\text{ELSE Send(}\{w\}, \text{AddOrig}(m, \{v\} \cup (S(v) \setminus \{w\}))\) \\
\} \\
\text{IF } R(v) \neq \emptyset \text{ THEN } \text{Store}(v, v^{\text{orig}}, \text{Context}(m)) \\
\text{ELSE Send(}\{v^{\text{orig}}\}, m) \\
\}$$

$$\text{ELSE}$$

$$\text{IF } v^{\text{orig}} \in \text{RetrieveExpectedReplies}(v, \text{Context}(m)) \text{ THEN}$$

$$\{ \\
\text{AddReply}(v, \text{Context}(m), m) \\
\text{RemoveExpectedReply}(v, v^{\text{orig}}, \text{Context}(m)) \\
\text{IF RetrieveExpectedReplies}(v, c) = \emptyset \text{ THEN} \\
\text{Send(Retrieve}(v, \text{Context}(m)), \text{GetReply}(v, \text{Context}(m))) \\
\}$$

The no knowledge propagation approach can be described by the following rules that hold for each middle-agent:

1. Registration, unregistration or modification messages are handled locally.
2. If a search request can be satisfied using the locally stored knowledge then reply to the requester with the result.
3. If a search request cannot be satisfied using the locally stored knowledge then it is routed via the full retrieval message routing (if the set of receivers is non empty); otherwise, reply to the requester with an unsuccessful result.

4. Store each result of a search request.

5. When all results of the search request are stored, assemble the results of the search request into a reply and send the reply back to the original requester (stored in $v^{\text{orig}}$ under the same context as the result).

There is no communication among middle-agents during the registration phase, but there is much more communication during the search for information and the update of information. Since there is no clue where to search for any information, the searching process must be exhaustive. All middle-agents, both the ones upstream at the top of the hierarchy and the ones downstream in the lower levels of the hierarchy, must be searched.

The requester can, however, limit the search space. The information about the depth of search can be added to the request (for instance specified by FIPA). The value of the search depth parameter decreases during the request propagation, resulting in a search within the specified boundary. The search depth limit can be described by additional rules that hold for each middle-agent in the hierarchy:

6. The forward of the search request is applicable only if the search depth parameter supplied in the search request is greater than zero. Otherwise reply to the requester with the result of local search.

7. If the search request is forwarded, decrease the value of the search depth parameter within the search request by one.

The requester can also limit the number of search results in a similar way. Using these search limitations the requester will query providers with the search request only within the specified boundary.

Since by using the no knowledge propagation only a local middle-agent knows the social information about an end-agent, the static impact of a middle-agent failure is higher than if other methods are used. Assume a graph $G$ of type M-E. The average redundancy and minimal redundancy are obviously equal to 1.

$$AR(G) = MR(G) = \frac{1}{N} \sum_{v \in V_G} d_G(v) = \frac{1}{N} \sum_{v \in V_G} 1 = 1$$

Social knowledge about an end-agent is propagated using 1 message (if we don’t count a possible acknowledgement message). Social knowledge is searched via the full retrieval message routing; therefore, it uses $2N$ messages (if the search is not limited), where $N$ is the number of middle-agents.
8.1.3.4 Knowledge Propagation on Demand

Both depth knowledge propagation and no knowledge propagation can be further improved with knowledge propagation on demand. Using this technique, information is discovered on demand and remembered for further use.

Knowledge propagation on demand can be described by the following additional rule that holds for each middle-agent in the hierarchy:

1. During the forwarding of the result of a search request remember the information that is contained in the result of the search.

Suppose a middle-agent needs to contact the parent middle-agent to search for information. When a response propagates back with possibly a positive result, middle-agents remember this information along the propagation path (see Figure 22). Assume that E-Agent3 and subsequently also E-Agent2 are searching for the capability that only E-Agent1 has. The number of search results is limited to 1. In the depth knowledge propagation method the capability of E-Agent1 is propagated using the root message routing from M-A 4 through M-A 2 to M-A 1.

![Figure 22: Knowledge propagation on demand applied to the depth knowledge propagation](image)

- E-Agent3 sends a search request to its local middle-agent (M-A 7).
- M-A 7 forwards the search request to M-A 3 since the search request cannot be satisfied locally.
- M-A 3 forwards the search request to M-A 1 since the search request cannot be satisfied locally.
- M-A 1 is able to satisfy the search request and replies back to M-A 3 with the search result.
- M-A 3 propagates the search result back to M-A 7 and remembers the information from the search result, i.e., the information about the registration of E-Agent1.
- M-A 7 finally propagates the search result to E-Agent3 and remembers the information from the search result.
- E-Agent2 sends a search request to its local middle-agent (M-A 6).
8. M-A 6 forwards the search request to M-A 3 since the search request cannot be satisfied locally.

9. M-A 3 is immediately able to satisfy the search request since the registration information has been remembered from the previous search, and replies back to M-A 6 with the search result since the parent retrieval message routing is used and all objectives of the message have been satisfied.

10. M-A 6 finally propagates the search result to E-Agent2 and remembers the information from the search result.

Propagation on demand brings one complication to social knowledge update. To assure that information gathered on demand is up-to-date, we must introduce one of the refresh mechanisms.

- **Subscribe and advertise mechanism.** When a middle-agent remembers some social knowledge, it subscribes for the update of this knowledge with the middle-agent that supplied the knowledge. When this knowledge gets updated then all middle-agents on the path of this update also send updates to all of their subscribers of this knowledge. This mechanism is used, for instance, in the KQML specification (see section 4.1.1).

- **Time stamping mechanism.** When a middle-agent stores social knowledge gathered on demand, this knowledge is time-stamped. The middle-agent can then examine the time-stamp to determine whether this knowledge is still valid or too old to be used. The examination process happens either periodically or at the time when this knowledge is accessed. Time stamping is a well known mechanism used, for instance, for revocation of certificates [82].

The no knowledge propagation approach can also be enhanced by propagation on demand. When the search mechanism returns successfully, the returned knowledge is remembered for further use along the path during the back propagation (see Figure 23). The subscribe-and-advertise or time stamping mechanism has to be used again to ensure that the knowledge is up-to-date. Assume that E-Agent3 is searching for the capability that only E-Agent1 has. The number of search results is limited to 1.
Figure 23: Knowledge propagation on demand applied to the no knowledge propagation

1. E-Agent3 sends a search request to its local middle-agent (M-A 7).
2. M-A 7 forwards the search request to M-A 3 since the search request cannot be satisfied locally.
3. M-A 3 forwards the search request simultaneously to M-A 1 and to M-A 6 since again the search request cannot be satisfied locally.
4. M-A 1 forwards the search request to M-A 2 since the search request cannot be satisfied locally.
5. M-A 2 forwards the search request simultaneously to M-A 4 and to M-A 5 since the search request cannot be satisfied locally.
6. M-A 4 is able to satisfy the search request and replies successfully back to M-A 2.
7. M-A 2 propagates the search result back to M-A 1 and remembers the information from the search result, i.e., the registration information about Agent1.
8. M-A 1 propagates the search result back to M-A 3 and remembers the information from the search result.
9. M-A 3 propagates the search result back to M-A 7 and remembers the information from the search result.
10. M-A 7 finally propagates the search result to E-Agent3 and remembers the information from the search result.

8.1.3.5 Knowledge Caching

Both depth knowledge propagation and no knowledge propagation can be further improved by using the knowledge caching mechanism.

The knowledge caching mechanism can be described by the following additional rule that holds for each middle-agent in the hierarchy:

1. During the forwarding of the search result only remember the knowledge that is contained in the result if the receiver is the original requester of the search, i.e., the receiver is not a middle-agent.

Knowledge caching is an alternative approach to knowledge propagation on demand in which knowledge is not remembered all the way back to the requester,
but only at the last middle-agent on the path to the requester. In this way the knowledge redundancy is very low despite the fact that knowledge is located at the proper places. Assume again that E-Agent3 is searching for the capability that only E-Agent1 has.

![Figure 24: Knowledge caching example](image)

1. E-Agent3 sends a search request to its local middle-agent (M-A 7).
2. M-A 7 forwards the search request to M-A 3 since the search request cannot be satisfied locally.
3. M-A 3 forwards the search request simultaneously to M-A 1 and to M-A 6 since the search request cannot be satisfied locally.
4. M-A 1 forwards the search request to M-A 2 since the search request cannot be satisfied locally.
5. M-A 2 forwards the search request simultaneously to M-A 4 and to M-A 5 since the search request cannot be satisfied locally.
6. M-A 4 is able to satisfy the search request and replies back to M-A 2 with the search result.
10. M-A 7 finally propagates the search result to E-Agent3 and remembers the information from the search result, i.e., the registration information about E-Agent1.

Note that we omitted describing all the cases in which the search was unsuccessful. The subscribe-and-advertise or time stamping mechanism has to be used again to ensure that the registration information is up-to-date.

All of these techniques are intended to work behind the scenes as part of the agent platform functionality. The hierarchy of middle-agents should be transparent to the end-agents in the system. The end-agents register and modify information using their local middle-agent and ask for information again from their local middle-agent. How this information propagates from one middle-agent to another is an agent platform issue only.

In this chapter the breadth, depth, and no knowledge propagation methods have been introduced. All are applicable to hierarchical types of social knowledge.
distribution including the DHT architecture. To test and evaluate these methods is a subject of the next chapter. Also in the next chapter we test the robustness of the DHT architecture to show how this architecture is able to deal in practice with a series of middle-agent failures.

8.1.4 Scalability of Dynamic Hierarchical Teams

Although the term scalability has been frequently used it has not been precisely defined. “Scalability has no commonly accepted precise definition” [87]. Researchers in the parallel processing community have been using Amdahl’s Law and Gustafson’s Law [35] and therefore tie notions of scalability to notions of speedup. It has been shown that these two are in fact identical [118]. Another definition of scalability is that “scalability of a machine for a given algorithm is the ratio of the asymptotic speedups on the real machine and on the ideal realization of an EREW PRAM (exclusive read/write parallel random access machine)” [87]. Nevertheless, speedup is not the main concern in the area of social knowledge since there is not one task that is split and solved in parallel. Therefore these definitions are not sufficient, and we present several other definitions.

“Scalability is measured by scaling up some resources while others remain untouched” [54].

“A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity” [83].

“Scalability is a function of parallel degradation where the source of this degradation is mainly due to the time spent in data exchanges between processors” [86].

“A scalable parallel processing platform is a computer architecture that is composed of computing elements. New computing element can be added to the scalable parallel processing platform at a fixed incremental cost” [63].

A formal definition of scalability in distributed applications is given in [121]. Very roughly, an application A is scalable in an attribute a if A can accommodate a growth of a up to defined maximum, if it is possible to compensate a performance degradation caused by increase of a, and if the costs that compensate performance degradation are limited. An important aspect of this definition is the distinction between performance and extensibility since the previous definitions are based on just one of these attributes.

To determine scalability without measuring resulting performance we can use definitions that are based on the extensibility and evaluate whether the cost to add a new computing element is fixed. Thus we reformulate the scalability definition that is based on extensibility [63] to be used in the area of social knowledge architectures.
**Definition 8.8** Let \( G = (V(G), E(G)) \) be a graph that represents the structure of middle-agents and assume that \( G \) is of type \( T \). Then let \( H \) be a graph of type \( T \) such that \( V(H) = V(G) \cup V^\delta(H) \) and \( E(H) = E(G) \cup E^\delta(H) \) where \( V^\delta(H) \) is a set of vertices that were added to \( V(G) \) and \( E^\delta(H) \) is a set of edges where each edge is adjacent to at least one vertex from \( V^\delta(H) \). If for each such \( G \) there exists \( \varepsilon > 0 \) such that for each \( V^\delta(H) \) there is \( E^\delta(H) \) with \( |E^\delta(H)| \leq \varepsilon |V^\delta(H)| \) then \( G \) is called **fixed scalable**.

**Theorem 8.3** If the graph \( G \) is DHT then \( G \) is **fixed scalable**.

**Proof.** Assume that \( G' \subset G \) is such team of \( G \) where its order \( \lambda \) is the biggest one. Assume that \( H \supset G \) such that \( V(H) = V(G) \cup \{v\} \). Then a set of edges \( E^\delta(H) \) has to satisfy for instance that \( \forall w(w \in V(G') \leftrightarrow \{v, w\} \in E^\delta(H)) \) to ensure that \( H \) is also DHT. Therefore \( |E^\delta(H)| = \lambda \). We can repeat this process for all \( v \in V^\delta(H') \) where \( H' \supset G \) where \( H' \) is again DHT, and where \( V(H') = V(G) \cup V^\delta(H') \) and \( E(H') = E(G) \cup E^\delta(H') \) hold. Therefore for each \( V^\delta(H') \) exists \( E^\delta(H') \) such that \( |E^\delta(H')| = \lambda |V^\delta(H')| \). □

Note that, for instance, the centralized architecture is obviously not fixed scalable since it cannot accommodate more than one vertex. Also, for instance, the teamwork-based technique (see section 5.3.2) is not fixed scalable since adding successive vertices to the system require more and more edges as the system grows in size. Therefore we list in Table 2 the fixed scalability for various structures of social knowledge distribution (see section 5).

**Table 2: Fixed scalability of various types of social knowledge distributions**

<table>
<thead>
<tr>
<th>Social knowledge distribution</th>
<th>Fixed scalable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>No</td>
</tr>
<tr>
<td>Distributed</td>
<td></td>
</tr>
<tr>
<td>Acquaintance models</td>
<td>No*</td>
</tr>
<tr>
<td>Architecture without middle-agent</td>
<td>No*</td>
</tr>
<tr>
<td>Hybrid</td>
<td></td>
</tr>
<tr>
<td>Facilitators</td>
<td>No†</td>
</tr>
<tr>
<td>Teamwork-based technique</td>
<td>No</td>
</tr>
<tr>
<td>Distributed matchmaking</td>
<td>Yes</td>
</tr>
<tr>
<td>Distributed blackboards</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic hierarchical teams</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* Note that end-agents are used in this case instead of middle-agents. Also note that if we ensure that each end-agent has a limited number of connections to other end-agents than these structures are fixed scalable.

---

16 The type \( T \) is for instance DHT, structure defined by distributed matchmaking, teamwork-based, etc.

17 The set of edges \( E^\delta(H) \) is added to \( V(G) \) after \( V^\delta(H) \) to ensure that the graph \( H \) is also of type \( T \).
8.1.5 Reconfiguration in Dynamic Hierarchical Teams

In Figure 18 and Figure 19, we present the concept of primary and secondary communication channels. During normal operation of the DHT architecture all middle-agents use only primary communication channels. Secondary communication channels are used when at least one of the following occurs:

- primary communication channels failed to transmit messages; or
- a receiving middle-agent failed (an observable failure).

The secondary communication channel does not mean ‘second’; there can be more than one secondary communication channel per primary communication channel. When a middle-agent is unable to use the primary communication channel then one of the secondary communication channels is used instead. The structure of interconnections of the primary communication channels is dynamically reshaped by this change.

![Dynamic reconfiguration of the structure of middle-agents as reaction to a failure of a middle-agent](image)

In Figure 25 we show how the secondary communication channels are used to dynamically reconfigure the structure of middle-agents. In this example M-A 1 failed and both M-A 11 and M-A 12 will eventually use their secondary communication channels to recover from this failure. The secondary communication channels that are used for the recovery now become primary.

The same process of dynamic reconfiguration is used when a communication channel fails to transmit a message. In this case also the sender middle-agent will
use the secondary communication channel. Note that the secondary communication channels replace only the primary communication channels that failed.

Another type of dynamic reconfiguration occurs when a new (or possibly cloned) middle-agent tries to register into the system or a previously failed middle-agent tries to reregister into the system. First, the middle-agent has to contact the already existing structure of middle-agents. There are several possibilities for creating the first communication channel. The middle-agent must either:

- broadcast to obtain the list of possible middle-agents that can be contacted or
- use the predefined address (or addresses) of a middle-agent that should be contacted or
- use AMS (white pages) agent to obtain the list of possible middle-agents that can be contacted or
- use other services of an agent environment to obtain the address (or addresses) of a middle-agent that should be contacted.

The middle-agent then uses the first communication channel (or channels) to connect to the already existing structure of middle-agents. Next, the middle-agent uses the first channel to discover the remaining primary communication channels and possibly also all secondary communication channels.

8.1.6 Failure Detection Mechanisms in Dynamic Hierarchical Teams

We described the mechanism of dynamic reconfiguration that is built on the assumption that the system is able to detect a failure of a primary communication channel or a failure of a middle-agent. Only upon a successful failure detection process can the system trigger the dynamic reconfiguration process. Without the failure detection of middle-agents a multi-agent system can possibly act as if a missing capability failure occurred (see section 7.2).

We do not consider the case in which an agent is able to detect a failure locally and act accordingly, for instance, to notify another agent in the community that the agent failed. Instead, we focus on the detection from outside the agent. The following mechanisms can be used to detect failure of a communication channel or of a middle-agent.

8.1.6.1 Response Timeout Mechanism

The no response and delayed response failures (see section 7.1) of a target agent or the failure of a communication channel (for instance a hardware failure, see section 7.2) can be detected by setting up a response timeout. This timeout is set by the originating agent and is used to ensure that the response arrives from the target agent back to the originating agent in a given timeframe. An assumption of this approach is that the target agent is expected to return the response to the originating agent.
The response timeout mechanism works as follows. The originating agent sends a message to the target agent and sets up the response timer with a specific timeout. The response timer has to be supplemented by the context of the conversation to distinguish among different replies in the general case that there are many conversations occurring simultaneously.

There are three possibilities for what can occur after the message has been sent to the target agent:

1. The response that is related to the context of the response timer arrived prior to the expiration of the timer. In this case, the response timer is discarded.

2. The response timer expired prior to the corresponding response arriving. The originating agent announces the detection of a no response failure (or of a delayed response failure) of the target agent or possibly of the communication channel. Also the incorrect knowledge failure has to be considered since there is a possibility that the target agent stored incorrect information about context into the response message.

3. The originating agent fails for some reason prior to the expiration of the timer and prior to the corresponding response arriving. The response timer is discarded and the conversation is reinitiated if the originating agent is unable to determine whether the failure affected the conversation with the target agent.

### 8.1.6.2 Heartbeat Mechanism

The no response failure (see section 7.1) of a target agent or the failure of a communication channel (for instance a hardware failure, see section 7.2) can be detected via periodic checking that the agent did not fail, called a heartbeat or keep alive mechanism [1]. Assume that an agent that is observed for possible failure is called the subject and an agent that is taking care of the subject is called the observer. The heartbeat mechanism can be either substantive or enforced.

- **Substantive** heartbeat - The observer initiates the heartbeat mechanism with the subject first. Then the subject periodically sends a status to the observer. Upon each incoming status the observer sets up a timer or resets an already running timer to an initial value. If the timer in the observer expires the subject is declared as having failed (no response or delayed response failure).

- **Enforced** heartbeat - The observer periodically sends a request for a report to the subject. Whenever the subject receives the request, the subject reports back status to the observer. If the observer did not receive the report within a specified timeframe, the subject is declared as having failed (no response or delayed response failure).

Both types of heartbeat mechanisms have several advantages in comparison to the response timeout mechanism.
1. There is a guarantee that a failed agent is discovered within the specified timeframe (if we do not consider a possibility that the observer can also fail).

2. The timeframe for guaranteed detection of a failure can be set up precisely according to a user specification.

   The disadvantages of the heartbeat mechanism are:

1. The communication and computation overhead for heartbeating messages.

2. The failure detection is restricted only to the no response failure (see section 7.1).

Moreover, both the heartbeat and the response timeout mechanisms can be used simultaneously to increase the possibility of detecting a failure.

### 8.1.6.3 Meta-Agent Observation

The meta-agent [101] can be used to detect all types of observable failures (see section 7.1) of an agent. The meta-agent observes the communication among agents and can use a reasoning process to discover possible failures of agents. Moreover, other types of multi-agent system monitoring techniques (see section 6.3.2) such as the SAM or the sentinel approach can be used for similar purposes.

An advantage to using a meta-agent is that any type of observable failure can be discovered using its observation and reasoning. A disadvantage is that the meta-agent has to monitor the communication among agents, an activity that can consume much time and processing power and can also violate the privacy of the agents.
9 Experimental Part

To determine advantages and disadvantages of the knowledge propagation methods (see section 8.1.3) and to test the robustness of the DHT architecture (see section 8.1) we use a real multi-agent system and we carry out several types of experiments on this system.

- The first goal is to compare the breadth, depth, and no knowledge propagation methods. The number of messages, total running time, and communication frequency are used for the comparison.

- The second goal is to test the robustness of the DHT architecture. A series of middle-agent failures are introduced into the system to test how the architecture is able to overcome the failures.

- The third goal is to compare various types of social knowledge distributions for robustness. The failure impact attributes (see section 7.3) are used for the comparison.

9.1 Comparison of Knowledge Propagation Methods

To determine which of the knowledge propagation methods presented in section 8.1.3 can be effectively used in practical applications, we performed testing on a real multi-agent system.

The test case environment is as follows. There are five middle-agents that have one shared global middle-agent (see Figure 26). There are twelve possible types of end-agents from multi-agent system used to control the chilled water part of a shipboard automation for US-Navy vessels [75]. An end-agent that is registered...
into the system has one or two from the seven possible capabilities. An end-agent location is randomly chosen, i.e., an end-agent registers with one of the five local middle-agents. All middle-agents and end-agents run on a single computer Pentium III/800Mhz as separate threads under Agent-OS [73]. The test case consists of an initial phase and a test phase described as follows:

1. During the initial phase 20 randomly chosen end-agents are created and registered into the system.

2. The test phase consists of 1000 actions. Three types of actions can occur.
   a) Create a new end-agent. A new end-agent of a randomly chosen type is created and registered to one of the middle-agents that is again chosen randomly.
   b) Delete one of the existing end-agents. One randomly chosen end-agent is unregistered from its local middle-agent and then the end-agent is deleted.
   c) Search for a capability. One randomly chosen end-agent that exists in the system sends a search request to its local middle-agent for a randomly chosen capability from six capabilities.

The distribution of probability to choose an action is as follows. The probability that the search action is chosen, denoted as $P_S$, is used as a parameter for each test case. The create and delete actions each has in every test case the same probability to be chosen, i.e., $(1-P_S)/2$.

The goal is to compare the breadth, depth, and no knowledge propagation methods. The number of messages (section 9.1.1), total running time (section 9.1.2), and communication frequency (section 9.1.3) are used for the comparison.

9.1.1 Comparison by the Number of Messages

The purpose of this test case is to determine which knowledge propagation method is the best one to be used when considering the number of messages, i.e., which method needs fewer messages for a completion of the test case.

---

18 Only one type of end-agent has two capabilities.

19 A uniform distribution of probability is used whenever we use the term randomly chosen.
Figure 27: Test case 1 of knowledge propagation methods for the number of messages

The X-axis in Figure 27 represents the probability that the search action is chosen, denoted as $P_S$. The Y-axis represents the total number of messages in the test run, where only messages that are related to the registration, unregistration, and to the search process are considered. Each value of each graph is the average (arithmetic mean) of 50 measurements.

From the measurements presented in Figure 27 we can conclude that:

- The no knowledge propagation method gives the best results in the case in which the probability that an agent uses the search action (instead of register or unregister action) is less than 35%.
- The depth knowledge propagation method gives the best results in the case in which $P_S$ is greater than 35% and less than 82%.
- The breadth knowledge propagation method gives the best results when $P_S$ is greater than 82%.
- The depth knowledge propagation method is nearly independent of the search probability parameter since the average deviation is 276, whereas
values measured for the other two methods have the average deviation greater than 1642.

Note that our goal is not to find the exact crossover points at which one method becomes better than another. These points are dependent on the structure of the middle-agents, on the number of capabilities that are used for the search, on the starting number of agents, and on other parameters.

To understand better why we obtained these results, we closely examine all possible message patterns that can occur during the test run.

1. The no knowledge propagation method has the following patterns:

   a. Register/unregister - 2 messages. An agent sends the request for registration to the local middle-agent and obtains the reply.

   b. Search - 2 or 12 messages. An agent sends the search request to its local middle-agent (see Figure 28).

   ![Figure 28: Message flow example for the search in the no knowledge propagation method](image)

   If the local middle-agent finds at least one agent that has the requested capability then the local middle-agent immediately sends a reply back to the requesting agent. In this case only two messages are used.

   If the local middle-agent is unable to find an agent with the requested capability then the local middle-agent (LocalDF3 in Figure 28) forwards the search request to all other middle-agents that are connected to it, in our case, just the global middle-agent. The global middle-agent always
forwards the search request to all remaining local middle-agents, since
the global middle-agent using the no knowledge propagation method
without caching and without propagation on demand, does not have any
agents registered locally. Figure 29 depicts an example as a workflow
diagram ([101] and [102]), where Agent18154 searches for
ManageSystemOperations capability in its local DF agent LocalDF3 and
all subsequent requests. After all responses propagate back to the original
requester there are 12 messages total.

Figure 29: Example of search using the no knowledge propagation method

2. The depth knowledge propagation method has the following patterns:

a. Register/unregister - 4 messages. An agent sends the request for
registration to the local middle-agent. The local middle-agent
forwards this registration request up the hierarchy to the global
middle-agent. When the local middle-agent obtains the reply back
from the global middle-agent, the local middle-agent propagates
the reply to the original requester.

b. Search - 2 or 4 messages. An agent sends the search request to
the local middle-agent.

If the local middle-agent finds at least one agent that has
the requested capability then the local middle-agent immediately sends
a reply back to the requesting agent. In this case only two messages are
used.

If the local middle-agent is unable to find an agent with the requested
capability then the local middle-agent forwards the search request to
the global middle-agent. With the back propagation of the reply there are
4 messages (see Figure 30).

Figure 30: Message flow example for the search in the depth knowledge
propagation method
3. The breadth knowledge propagation method has the following patterns:

   a. Register/unregister - 8 messages. An agent sends the request for registration to the local middle-agent and obtains a reply. The local middle-agent forwards the register request to the global middle-agent and obtains a reply. The global middle-agent forwards the register request to all remaining local middle-agents.

   b. Search - 2 messages. An agent sends the request for search to the local middle-agent and obtains only one reply back.

Using all possible message patterns we can compute the functional dependence of the number of messages on the search probability. Assume that $P_S$ denotes the search probability and $P_F$ denotes the probability that an agent searches for a capability and the result of the search is found by the local middle-agent. $A$ denotes the number of actions and $B$ is the number of initially registered agents. $R$ denotes the number of messages that are needed to register an agent, $S$ denotes the number of messages that are needed to search only locally, $T$ denotes the number of messages that are needed to search globally, and $N$ denotes the total number of messages. Then following equations hold.

$$N = A \cdot (R \cdot (1 - P_S) + P_S \cdot (S \cdot P_F + T \cdot (1 - P_F))) + B \cdot R \quad (8)$$

$$= R \cdot (A + B) + P_F \cdot A \cdot (T - R + P_F \cdot (S - T)) \quad (9)$$

Assume for the given test case that $P_F$ is a constant value. Then Equation 9 is the equation of a line since only $P_S$ is a variable. This analysis explains why the test results from Figure 27 are close to a linear dependence on $P_S$.

If we substitute all the constants except $P_F$ with their actual values from the test case, we obtain the following equations.

$$N_{N_0} = 2040 + 10000 \cdot (1 - P_F) \cdot P_S \quad (10)$$

$$N_{Depth} = 4080 - 2000 \cdot P_F \cdot P_S \quad (11)$$

$$N_{Breadth} = 8160 - 6000 \cdot P_S \quad (12)$$

Based on these equations, we can draw the areas that represent the dependency of each method not only on $P_S$, but also on $P_F$ (see Figure 31). Note that $N_{Breadth}$ is not dependent on $P_F$ (see Equation 12), which means that the graph that represents this method is not depicted as an area, but as a line ($P_S$ is limited to the range $[0,1]$). This characteristic corresponds with the fact that the breadth knowledge propagation method only searches locally.
To determine the exact value of $P_F$ theoretically, the equation must consider all possible variations of agent registrations to all local middle-agents, i.e., discrete probability distribution $P_E(x)$ that consists of probabilities of how many end-agents, denoted as $x$, are simultaneously registered to one middle-agent (see Figure 32 for $x \leq 10$).

![Figure 31: Theoretical comparison of knowledge propagation methods](image1)

Next step is to multiply this probability distribution by the distribution of probability that the group of end-agents of given size is selected (e.g., when a middle-agent does not have any end-agent registered then the probability that this middle-agent will be asked to search for a capability is equal to 0) to obtain $P_S(x)$. Then we multiply this distribution by probabilities $P_x$ to find a capability only

![Figure 32: Probability distributions and probabilities used to compute $P_F$](image2)
locally for given number of end-agents registered to one middle-agent. Finally, we sum these values to obtain $P_F$ approximately equal to 0.51. This value has been verified also by the simulation that uses random number generators to reflect the behavior of this multi-agent system.

From the experimental results of the depth and no knowledge propagation methods we can compute, using the least squares approximation, that $P_F$ is approximately equal to 0.51 for this test case, i.e., it reflects the theoretically computed value.

9.1.2 Comparison by Total Running Time

Another way to compare the knowledge propagation methods is to measure the total running time that is needed for the completion of the test case. The goal is to measure the dependency of the total running time on the probability that a search action is chosen, $P_S$. The same test environment as in the previous test case is used.

![Figure 33: Test case 1 of knowledge propagation methods for the total running time](image)
The X-axis in Figure 33 represents the probability of choosing the search action, $P_S$. The Y-axis represents total running time of the test run in seconds. Each value of each graph is the average (arithmetic mean) of 50 measurements.

From the measurements presented in Figure 33 we can conclude that:

- The no knowledge propagation method gives the best results in cases where the probability that an agent uses the search action is less than 22%.
- The depth knowledge propagation method gives the best results in cases where $P_S$ is in the range of 22% to 68%.
- The breadth knowledge propagation method gives the best results in cases where $P_S$ is greater than 68%.
- All three crossing points on the knowledge propagation graphs are at similar positions as in Figure 27, although the shapes of the dependency on $P_S$ are different. On the left side of the graph the probability that an agent is created is higher (50%) than on the right side of the graph (0%). The left sides of the graphs are significantly elevated since an agent has a very long creation time due to the full initialization of its planning templates.

To compare the measurements of the run-time duration and the number of messages we measure the percent error\(^\text{20}\) of each knowledge propagation method (see Figure 34).

\(^{20}\) The percent error is computed from the mean value and from the average deviation.
From the measurements presented in Figure 34 we can conclude that:

- The percent errors for the number of messages for the breadth and depth knowledge propagation methods are greater than the percent errors for run-time duration for $P_s$ greater (or equal) 0.6, except for breadth knowledge propagation if $P_s = 1$ since the number of messages is in this case a constant value.

- The percent errors for the number of messages for the no knowledge propagation method are greater than the percent errors for run-time duration except $P_s = 0$ since the number of messages is a constant value.
• The average of all percent errors for the number of messages is 6.0 and the average for run-time duration is 6.2.

From these results we can conclude that the measurements of the number of messages (see Figure 27) give overall the same precise comparison of the different knowledge propagation methods in this testing environment as the measurements of the run-time duration (see Figure 33). Nevertheless, in the area of $P_s > 0.6$ it is more precise to measure the run-time duration in this testing environment.

9.1.3 Comparison by Communication Frequency

Another way to compare different types of knowledge propagation methods is to observe the communication frequency in the multi-agent system, i.e., the number of messages per second. Ten test runs for each of the knowledge propagation methods from are therefore observed for their communication frequency.

Figure 35 depicts the communication frequency for the breadth, depth, and no knowledge propagation methods. The test runs (10 for each method, Y-axis) are split into time slots of 1 second duration, X-axis, starting from the beginning of the test run. For a given timeslot, the Z-axis represents the number of messages in the whole multi-agent system. The probability of choosing the search action, $P_s$, is equal to 0.5 for this test case since this is a compromise between searching and registering.

The graph of communication frequency for the depth knowledge propagation method is smooth in comparison with the breadth and no knowledge propagation methods. The reason is that for the depth knowledge propagation the requests are handled by low number of middle agents, i.e., either by two middle-agents for registration and unregistration or by one or two middle-agents for the search requests. For the breadth and no knowledge propagation either one middle-agent or all six middle-agents are used.
Figure 35: Communication frequency of knowledge propagation methods for $P_S = 0.5$
The comparison of the knowledge propagation methods by the communication frequency in the whole range of the probability of choosing the search action, $P_S$, is shown in Figure 36. The percent errors, i.e., the average deviation divided by mean value, of the first 100 seconds of each test run for each knowledge propagation method are presented.

![Figure 36: The percent errors of knowledge propagation methods](image)

From the measurements presented in Figure 36 we can conclude that the communication frequency of the depth knowledge propagation method is smooth in comparison with the breadth and no knowledge propagation methods since the percent errors are lower, except the point in which $P_S = 0$. This exception appears since all requests of end-agents are handled locally by one middle-agent and therefore constantly two messages are used for one request. Thus the depth knowledge propagation method is overall more stable to the variety of end-agent requests then the other two methods in this test case.

9.2 Experiments with Robustness

The following set of experiments is targeted to test the robustness of the multi-agent system architecture. The first set of experiments is targeted to the DHT architecture and the second set is focused on comparison of the robustness of different types of multi-agent system architectures.

9.2.1 Robustness in the Dynamic Hierarchical Teams

To test the robustness of the DHT architecture (see section 8.1) in practical experiments, we first create an experiment that consists of teams with just one member, i.e., the pure hierarchical architecture. Then we increase the number of
members in the team to test how this change affects the robustness of the whole system.

### 9.2.1.1 Hierarchy without Redundancy

The test case setting is as follows. The testing architecture of middle-agents consists of six middle-agents that have one shared parent middle-agent, i.e., the pure hierarchical structure. The test case has the same initial phase and similar test phase as the experiment in section 9.1. The test phase in this case consists of 500 randomly chosen actions with the probability of choosing the search action equal to 0.9. After 100 actions, corresponding to 100 seconds, we simulate the no response failure (see section 7.2) of the global middle-agent and after 300 actions the no response failure gets resolved.

![Figure 37: One no response failure of the global middle-agent in the hierarchical architecture](image)

**Figure 37: One no response failure of the global middle-agent in the hierarchical architecture**

Figure 37 depicts the communication frequency for the hierarchical architecture filtered in such a way that only outgoing messages from the global middle-agent are considered. The architecture uses the depth knowledge propagation method (see section 8.1.3.2). The test run is split into time slots of 20 seconds.
(X-axis) and each column (Y-axis) represents the number of messages in a given time slot.

After 100 seconds the global middle-agent simulates the no response failure and the system is not able to use the global middle-agent to propagate requests. Thus only search requests that can be solved using local knowledge are processed now. All other requests stay pending and the local middle-agents periodically retry sending them to the global middle-agent.

After 300 seconds from the beginning of the test case the no response failure of the global middle-agent gets resolved and the global middle-agent has to resolve all pending requests plus all new requests. Therefore, the number of outgoing messages from the global middle-agent is temporarily increased approximately seven times relative to the state before the failure.

9.2.1.2 Redundant Hierarchy

In this test case we increased the number of members in the team to test how this change affects the robustness of the whole system. The test case setting is as follows. The testing architecture consists of three middle-agents that form the global team plus six middle-agents that are evenly distributed using their primary connections among the three global middle-agents (see Figure 38). The primary communication channels are represented by solid lines and the secondary ones by dashed lines.

![Figure 38: Testing architecture for a robustness testing](image)

The test case has the same initial phase and a similar test phase as in the previous experiment (see section 9.2.1.1). After 300 actions, corresponding to 150 seconds, we simulate the no response failure (see section 7.2) of the first global middle-agent and after the next 300 actions the no response failure of the second middle-agent. After 300 actions from the beginning of the test case the failure of the first middle-agent gets resolved and after the next 300 actions the failure of the second middle-agent also gets resolved.
Figure 39: Two no response failures of the global middle-agents in the DHT architecture with three members of the team

Figure 39 depicts the communication frequency for the DHT architecture. The architecture uses the depth knowledge propagation method (see section 8.1.3.2). The test run is split into time slots of 20 seconds (X-axis) with the number of messages in each time slot on the Z-axis connected by a line. The first graph on the Y-axis labeled GM-A1 is filtered in such a way that only outgoing messages from the first global middle-agent are considered. The second graph shows outgoing messages from the second global middle-agent and the third graph shows the last global middle-agent.

After 150 seconds the first global middle-agent simulates the no response failure and the system is not able to use the first global middle-agent to propagate requests. The requests that are sent to the first global middle-agent stay pending until the system discovers that the first global middle-agent failed. The system uses the heartbeat mechanism for the failure detection (see section 8.1.6.2) in this test case. The local middle-agents that are initially connected to the first global middle-agent dynamically switch to another global middle-agent, in this case the second middle-agent. Figure 39 indicates the dynamic switch to the second middle-agent by the increased communication frequency of the second middle-agent.

After 300 seconds from the beginning of the test case also the second global middle-agent simulates the no response failure. In this case also the local middle-agents that are initially connected to the first and to the second global middle-agent...
dynamically switch to the third global middle-agent and the system is still able to respond to the incoming requests.

After 450 seconds from the beginning of the test case the first global middle-agent is repaired, followed by the second middle-agent 150 seconds later. The local middle-agents dynamically switch back to their preferred global middle-agents and the system starts to use all global middle-agents again.

The test case shows that the DHT architecture with teams that consist of $N$ middle-agents (3 in this test case) is able to withstand at least $N - 1$ failures of the middle-agents, i.e., the maximal failure impact $FI_{N-1} = 0\%$ (see section 7.3).

### 9.3 Comparison of Social Knowledge Distributions

To compare the robustness of various types of social knowledge distributions, we selected several examples of centralized, distributed, pure hierarchical, and DHT architectures (see section 5). The goal is to evaluate these architectures based on the failure impact and on the average (AR) and minimal (MR) redundancy (see section 7.3), i.e., static impact of one or two simultaneous failures.

**Table 3: Failure impact and redundancy of various types of social knowledge distributions**

<table>
<thead>
<tr>
<th>Social knowledge distribution type</th>
<th>$FI_1$ [%]</th>
<th>$FI_1$ [%]</th>
<th>$FI_2$ [%]</th>
<th>$FI_2$ [%]</th>
<th>AR, MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>Distributed</td>
<td>40/100/N</td>
<td>60/100/N</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>$N$ equally distributed centers ($N &gt; 1$)</td>
<td>100/100/N</td>
<td>200/200/N</td>
<td>200/200/N</td>
<td>200/200/N</td>
<td>1</td>
</tr>
<tr>
<td>Hierarchical with 2 levels (or DTH-1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>$N$ equally distributed centers + 1 global center</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100/N</td>
<td>2</td>
</tr>
<tr>
<td>DHT with 1 team (2 levels)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$N$ equally distributed centers + 2 global centers*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$N$ global centers†</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$N$</td>
</tr>
<tr>
<td>Teamwork-based tech.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$N$</td>
</tr>
</tbody>
</table>

---

This means that the first center covers 40% of end-agents in the system and the second center remaining 60%.
* The depth knowledge propagation is used.

† The breadth knowledge propagation is used. Note that the description ‘$N$ global centers’ does not imply a team of size $N$. It only means that all $N$ middle-agents hold social knowledge about all end-agents in the system.

From the results presented in Table 3 we can conclude that:

1. Centralized social knowledge obviously does not have any robustness since any failure results in 100% failure impact, i.e., the whole system is affected.

2. Distributed social knowledge has lower failure impact than the centralized one, but it is not possible to obtain an impact of 0% since this knowledge is not redundant.

3. If there is at least one redundant global center that covers 100% of the social knowledge in the system (all end-agents), i.e., the hierarchical, DHT, and teamwork-based architectures, the system is tolerant to one failure and the failure impact for more than one failure is also lowered.

4. The DHT architecture that has at least two global centers is tolerant to at least two failures that occur simultaneously. When this architecture uses the breadth knowledge propagation method, it has the highest possible AR and MR.

5. The teamwork-based architecture has also the highest possible AR and MR, but this architecture does not offer the flexibility to decrease the number of connections among middle-agents to improve scalability.

9.4 Summary of Results

The first part of the experiments (see section 9.1) revealed that the breadth knowledge propagation method is the most efficient method in this testing environment when the probability $P_S$ that end-agents request a search action (instead of a register or unregister action) is close to 1. The no knowledge propagation method is the most efficient in the case in which end-agents register or unregister very frequently, i.e., when $P_S$ is close to 0. The depth knowledge propagation method is the most efficient in the other cases, i.e., when $P_S$ is near 0.6. The evaluation of the number of messages gives overall the same precise comparison as the total running time. The comparison by communication frequency shows that the depth knowledge propagation method is overall more stable to the variety of end-agent requests then the other two methods in this test case.

The robustness testing experiments (see section 9.2) show that the DHT architecture with teams that consist of $N$ middle-agents is able to withstand at least $N-1$ simultaneous failures of middle-agents (2 failures have been experimentally tested), i.e., the maximal failure impact $FI_{N-1} = 0\%$ (see section 7.3). That also means that this architecture gives the system designer the ability to create the architecture that has a predefined level of robustness.
The comparison of various social knowledge distributions (see section 9.3) confirmed that the centralized and distributed architectures have very low robustness (static impact) in comparison with the hierarchical, DHT, and teamwork-based technique architectures. The DHT architecture can be configured to have the same redundancy as the teamwork-based technique while preserving the fixed scalability.
10 Conclusions

This chapter presents the fulfillment of goals of this thesis, summarizes achieved results presented in this thesis, and outlines the contribution of the described research.

10.1 Fulfillment of Goals

- The dynamic hierarchical teams (DHT) structure that can be used to increase the robustness of the whole multi-agent system has been proposed and implemented. It has been proved that the structure that consists of teams of size $N$ is fault tolerant to failure of $N - 1$ middle-agents or $N - 1$ communication channels. Therefore, the DHT structure offers and ensures demanded level of fault tolerance. It has been proved that the DHT structure is maximally fault tolerant. It has been also proved that this structure is fixed scalable. Therefore, scalability is still preserved while robustness of the system increases.

- Several methods for social knowledge propagation in the DHT structure have been defined, such as breadth, depth, and no knowledge propagation. Also the knowledge propagation on demand and knowledge caching mechanisms that can improve the efficiency of the knowledge propagation methods have been proposed. Also the response timeout, heartbeat, and meta-agent observation failure detection mechanisms that can be used to detect possible failures of agents in the multi-agent system have been described.

- Proposed knowledge propagation methods have been experimentally tested to determine their advantages and disadvantages on a real multi-agent system. The experiments revealed that the breadth knowledge propagation method is the most efficient method in the presented testing environment when the probability $P_{S}$ that end-agents request the search action (instead of the register or unregister action) is close to 1. The no knowledge propagation method is the most efficient in the case that end-agents register or unregister very frequently, i.e., when $P_{S}$ is close to 0. The depth knowledge propagation method is the most efficient in other cases, i.e., when $P_{S}$ is near 0.6.

10.2 Summary of Results

Various types of multi-agent system architectures have been summarized from the point of view of social knowledge distribution and management. Static architectures, where social knowledge is statically inserted at design time into the community of agents, and the dynamic architectures that are open to changes in the system at run-time have been identified. The table of middle-agent roles has
been reviewed by adding several types of middle-agents and the scope of the classification has been also increased by adding new types of preferences. A missing type of middle-agent has been identified, called Job Agency, and the behavior for it has been proposed. Centralized, distributed, and hybrid types of social knowledge distribution have been identified and several improvements to already existing architectures have been suggested.

It has been summarized how different types of existing social knowledge management and distribution techniques are standardized and practically used. KQML and FIPA standards, LARKS, SDL, and DAML-S language specifications used for social knowledge encoding, and PHOSPHORUS and SLP service location mechanisms have been briefly described. Then the functionality of selected multi-agent system implementations has been described from the point of view of social knowledge management. JADE and FIPA-OS multi-agent system development frameworks that use FIPA compliant social knowledge management techniques have been briefly described and it has been shown how social knowledge is distributed and managed in systems such as ProPlanT, SHADE, COINS, and InfoSleuth.

The classification of agent failure types and of multi-agent system failure types from an observer’s point of view has been proposed. Middle-agent failure impact attributes, average redundancy, and weighted average redundancy measures that can be used to measure the fault tolerance or robustness of the multi-agent system have been also proposed. Additionally, techniques that increase the robustness of an agent or of a multi-agent system such as the warm and hot backup, N-version programming, multi-agent system monitoring, and multi-agent system architecture improvement techniques have been summarized.

The main contribution of this thesis is that the dynamic hierarchical teams (DHT) architecture that can be used to increase the robustness of the multi-agent system has been proposed and implemented. The DHT structure has been defined using graph theory and it has been proved that the structure which consists of teams of size \( N \) is fault tolerant to failure of \( N - 1 \) middle-agents or \( N - 1 \) communication channels. Moreover, it has been proved that the structure is maximally fault tolerant. It has been proved that the DHT structure is fixed scalable. Several methods for social knowledge propagation have been defined, such as breadth, depth, and no knowledge propagation. Also the knowledge propagation on demand and knowledge caching mechanisms that can improve the efficiency of the knowledge propagation methods have been proposed. The response timeout, heartbeat, and meta-agent observation failure detection mechanisms that can be used to detect possible failures of agents in the multi-agent system have been also described.

Proposed knowledge propagation methods have been experimentally tested to determine their advantages and disadvantages on a real multi-agent system. The experiments revealed that the breadth knowledge propagation method is the most efficient method in the presented testing environment when the probability \( P_S \) that end-agents request the search action (instead of the register or unregister action) is close to 1. The no knowledge propagation method is the most efficient in
106

the case that end-agents register or unregister very frequently, i.e., when \( P_S \) is close to 0. The depth knowledge propagation method is the most efficient in other cases, i.e., when \( P_S \) is near 0.6.

It has been experimentally proved that the proposed DHT architecture increases the robustness of the whole multi-agent system in comparison with the pure hierarchical architecture. Finally different types of social knowledge distributions have been evaluated based on the failure impact attributes and average redundancy measure. The comparison confirmed that the centralized and distributed types of architectures have very low robustness in comparison with the hierarchical and DHT architectures.

10.3 Contribution

The contribution of the described research can be seen mainly in the following areas.

- Social knowledge is one of the main essential components of a multi-agent system. Thus research in the area of robustness, scalability, and management of this knowledge is very important. The structure of social knowledge that offers predefined level of fault tolerance is demanded since the agents depend on this knowledge to be able to interact as expected. Development of the DHT structure of middle-agents offers fault tolerant and fixed scalable structure of social knowledge that fulfills these requirements.

- The DHT structure has been defined using graph theory and it has been proved that the structure which consists of teams of size \( N \) is fault tolerant to failure of \( N - 1 \) middle-agents or \( N - 1 \) communication channels. Moreover, it has been proved that the structure is maximally fault tolerant and fixed scalable.

- Several methods for social knowledge propagation in the DTH structure have been defined, such as breadth, depth, and no knowledge propagation. Also the knowledge propagation on demand and knowledge caching mechanisms that can improve the efficiency of the knowledge propagation methods have been proposed. Proposed knowledge propagation methods have been experimentally tested to determine their advantages and disadvantages on a real multi-agent system. The experiments revealed that all of these methods can be efficiently used under various conditions.

- Various types of multi-agent system architectures have been compared from the point of view of social knowledge distribution and management. A missing type of middle-agent has been identified, called Job Agency, and the behavior for it has been proposed. This middle-agent offers a location of preferences and capabilities that is not known to be used so far in the area of multi-agent systems.
11 Future Work

In this thesis we have concentrated on the development and implementation of various social knowledge management techniques in the area of multi-agent systems. Mainly we have focused on knowledge propagation mechanisms. Various types of middle-agents can use these mechanisms. Further research can concentrate on the practical comparison of various types of middle-agents (matchmaker, broker, recruiter, etc.) and on the comparison of techniques that use middle-agents with techniques that do not use middle-agents.

Another area that can be further investigated is social knowledge representation. We have presented various languages that are designed for social knowledge representation. Although these languages are a part of the standardization process, this area is still open to new ideas and development.

One of the contributions of this thesis is the classification of the failures either of agents or of the whole multi-agent system. We suggest for future work that this classification can be used in the area of intrusion detection. To practically test these ideas can also be the subject for future work.
Bibliography


[106] Pěchouček M., Mařík V., and Štěpánková O.: *Role of Acquaintance Models in Agent-Based Production Planning System*, In (Klusch M., Kerschberg L., eds.)


