Computing and Informatics, Vol. 22, 2003, 597–622

MIDDLE-AGENTS ORGANIZED IN FAULT TOLERANT AND FIXED SCALABLE STRUCTURE

Pavel TICHÝ

Rockwell Automation Research Center and Czech Technical University
Americká 22
120 00 Prague, Czech Republic
e-mail: ptichy@ra.rockwell.com

Revised manuscript received 23 March 2004

Abstract. Agents in a multi-agent system usually use middle-agents to locate service providers. Since one central middle-agent represents a single point of failure and communication bottleneck in the system, therefore a structure of middle-agents is used to overcome these issues. We designed and implemented a structure of middle-agents called dynamic hierarchical teams that has user-defined level of fault-tolerance and is moreover fixed scalable. We prove that the structure that has teams of size λ has vertex and edge connectivity equal to λ, i.e., the structure stays connected despite $\lambda - 1$ failures of middle-agents or $\lambda - 1$ communication channels. We focus on social knowledge management describing several methods that can be used for social knowledge propagation and search in this structure. We also test the fault-tolerance of this structure in practical experiments.

Keywords: Multi-agent systems, fault tolerance, scalability

1 INTRODUCTION

One of the main advantages in using multi-agent systems is fault tolerance. When an agent fails a multi-agent system could "offer" another agent that can be used instead. Is this enough to ensure fault tolerant behavior? If a multi-agent system uses a middle-agent [4] to search for the capabilities of providers, i.e., to search for an alternative agent with the same capability, then this middle-agent can become a single point of failure, i.e., social knowledge is centralized in this case. It is not possible to search for capabilities of other agents and to form virtual organizations

any more if the system looses the middle-agent. Fault tolerance issue is tightly coupled with load sharing. When only one middle-agent is used then it becomes a communication bottleneck and can be easily overloaded.

Social knowledge in 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, their location (address), their capabilities (services), the language they use, their actual state, their conversations, behavioral patterns, and so on [13]. Knowledge that is located in agents can be formally split into two distinct types [2]:

domain knowledge (or *problem-solving knowledge* [13]) – 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.

Several approaches have been used already to deal with fault tolerance and scalability issues. The teamwork-based technique [11] has been proposed that uses a group of N middle-agents where each middle-agent is connected to all other middle-agents forming a complete graph. This technique offers fault tolerance but since it uses a complete graph this structure is not fixed scalable as shown in Section 2.5. Another example is distributed matchmaking [18] that focuses on increasing the throughput and scalability of matchmaking services by creation of hierarchy but does not deal with fault tolerance. Another approach to distribute social knowledge among agents is to use for instance acquaintance models [13], but this technique implicitly does not ensure fixed scalability since in the worst case each agent keeps knowledge about all other agents (see Section 2.5).

2 DYNAMIC HIERARCHICAL TEAMS ARCHITECTURE

We propose the dynamic hierarchical teams (DHT) architecture 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.

2.1 DHT Architecture Description

Assume that a multi-agent system consists of middle-agents and other agents are referred to as end-agents. Middle-agents form a structure that can be described by the graph theory. Graph vertices represent middle-agents and graph edges represent a possibility for direct communication between two middle-agents, i.e., communication channels.

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.

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. Groups of these middle-agents are called *teams* (see Figure 1). Whenever one of the middle-agents from the team fails, other middle-agents from the team can subrogate this agent.

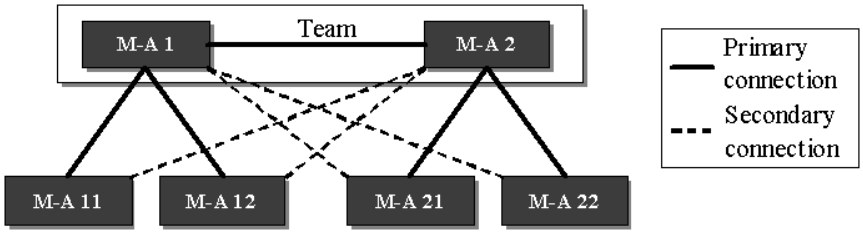


Fig. 1. Example of 2-level DHT architecture

During the normal operation of the DHT structure all middle-agents use only primary communication channels. The usage of secondary communication channels will be further described in Section 2.6.

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

More complex is the structure of middle-agents (not 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. The members of the top-most 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 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 full-connected network of middle-agents is created, i.e., a structure similar to the teamwork-based technique [11]. The DHT structure is therefore flexible in this respect.

Let G be a graph where each middle-agent i in the dynamic hierarchical teams (DHT) structure is represented by a graph vertex $v_i \in V$ and each primary or

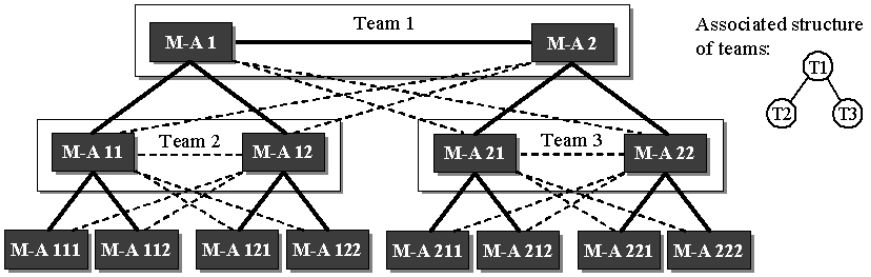


Fig. 2. Example of 3-level DHT architecture and associated structure of teams

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 1 (DHT graph)**.** A graph G will be called 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. $\forall v(v \in V(G) \setminus V_1 \rightarrow \exists j \forall w(w \in V_j \rightarrow \{v, w\} \in E(G)))$[1]
2. $\forall v(v \in V(G) \wedge v \notin V_1 \cup \ldots \cup V_n) \rightarrow \exists! j \forall w(w \notin V_j \rightarrow \{v, w\} \notin E(G)))$[2]
3. $\forall j((j > 1) \wedge (j \leq n) \rightarrow \exists! k((k < j) \wedge \forall v \forall w(v \in V_j \wedge w \in V_k \in \{v, w\} \in E(G)) \wedge \forall u \forall m(u \in V_m \wedge (m < j) \wedge (m \neq k) \rightarrow \{v, u\} \notin E(G)))$[3]

Definition 2 (DHT-λ graph)**.** The graph G is called DHT-λ if G is DHT and $|V_i| = \lambda$ for every $i = 1, \ldots, n$; where $\lambda \in \mathcal{N}$.

2.2 Fault Tolerance in DHT Architecture

The vertex and edge connectivity of a graph (see below) provide information about possibility to propagate knowledge through the network of middle-agents and communication channels despite failures on these multi-agent system components. Thus, only the dynamic impact of these failures is studied in this paper. Static impact of these failures is out of scope of this paper. In that case we study various static parameters, e.g., a minimal/maximal percentage of end-agents about which social knowledge stored in middle-agents is lost when k middle-agents fail simultaneously,

[1] For all vertices v of G except V_1 (since there is no team with lower index than V_1) there has to be a team such that v is connected to all members of this team.

[2] 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.

[3] All members of each team except G_1 are connected to all members of exactly one other team with lower index.

minimal/average redundancy of the social knowledge occurs, etc. Problems related to intrusions to the system, e.g., denial of service attack, disruption of connection, etc. are also out of the scope of this paper.

The fault tolerance of an undirected graph is measured by the vertex and edge connectivity of a graph [5]. To briefly summarize these terms, a graph G is said to be λ *vertex-connected* if the deletion of at most $\lambda-1$ vertices leaves the graph connected. The greatest integer λ such that G is λ vertex-connected is the *connectivity* $\kappa(G)$ of G. A graph is called λ *edge-connected* if the deletion of at most $\lambda - 1$ edges leaves the graph connected. The greatest integer λ such that G is λ edge-connected is the *edge-connectivity* $\lambda(G)$ of G.

Claim 1. If the graph G is DHT-λ then for each vertex $v \in V(G)$ there exists a vertex $w \in V_1$ such that there is a path in G starting from v and ending at w after removing $\lambda - 1$ vertices or $\lambda - 1$ edges from G.

Proof. Assume the case where $v \notin V_1$ since otherwise the path is v itself. For each team of DHT-λ $|V_j| = \lambda$. Thus there are at least λ edges $\{v, w_1\}$ such that $\exists j \forall w_1 (w_1 \in V_j \rightarrow \{v, w_1\} \in E(G))$. Since $|V_j| = \lambda$ then after the elimination of $\lambda - 1$ vertices or $\lambda - 1$ edges there exists a path starting from v and ending at w_1 where $w_1 \in V_j$. If $j = 1$ then the resulting path is vw_1. Otherwise, since $|V_j| = \lambda$ the rule 3) from the definition of DHT (definition 1) can be repeatedly applied to construct a path in G despite the elimination of $\lambda-1$ vertices or $\lambda-1$ edges starting from w_1 and ending at w_k where $w_k \in V_1$. Therefore in this case the resulting path is $vw_1w_2 \ldots w_k$. □

Lemma 1. If the graph G is DHT-λ then G is λ vertex-connected and λ edge-connected.

Proof.

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

it separates the graph at least into pieces C_1 and C_2. Let $v_1 \in V(C_1)$ and $v_2 \in V(C_2)$. We already proved that after removing $\lambda - 1$ vertices there exists a path starting from a vertex v_1 (or v_2) and ending at w_1 (or w_2) where $w_1 \in V_1$ and $w_2 \in V_1$. If $w_1 = w_2$ then a path from v_1 to v_2 already exists. Otherwise since G_1 is a complete graph then any two vertices $w_1 \neq w_2$ where $w_1 \in V(G_1)$ and $w_2 \in V(G_1)$ are connected after elimination of $\lambda - 1$ vertices from G. Thus there is no vertex cut set of size $\lambda - 1$.

□

Claim 2. If the graph G is DHT-λ then the minimum degree $\delta(G) = \lambda$.

Proof. We already proved that G is λ edge-connected and therefore $\delta(G) \geq \lambda$. From the definition of DHT rule 1) (definition 1) and the fact that $V_1 \cup \ldots \cup V_n \neq V(G)$ there has to be at least one vertex $v' \in V(G)$ for which $v' \notin V_1 \cup \ldots \cup V_n$ holds and $\exists j \forall w (w \in V_j \rightarrow \{v', w\} \in E(G))$. Since $|V_j| = \lambda$ for DHT-λ thus there are at least λ edges $\{v', w\}$ 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 λ edges $\{v', w\}$ and therefore $d(v') = \lambda$. Since $\delta(G) \geq \lambda$ and $d(v') = \lambda$ thus $\delta(G) = \lambda$. □

Theorem 1. If the graph G is DHT-λ 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-λ is λ vertex-connected and λ edge-connected thus 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 [5]. We already proved that $\delta(G) = \lambda$ therefore $\kappa(G) \leq \lambda$ and $\lambda(G) \leq \lambda$. □

The DHT structure where teams consist of λ 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 $\delta(G)$ [21].

Theorem 2. The graph G of type DHT-λ is maximally fault tolerant.

Proof. We already proved that the graph G of type DHT-λ has vertex connectivity $\kappa(G) = \lambda$ and we also proved that it has the minimum degree $\delta(G) = \lambda$. □

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

The problem of directly computing the most survivable deployment of agents has been introduced recently in [10]. Although, disconnect probabilities of all nodes have to be known either by collecting statistical data or by experts (that can be hard to obtain and values are not only application specific but can also vary in time); then it is possible to compute the optimal survivable deployment. This probabilistic approach cannot be used in our study since these values are not available.

In addition, N-version programming [3] 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.

2.3 Impact of Middle-Agent Failures

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:

Definition 3 (k-failure impact)**.**

Fi_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.

FI_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 Fi_1 and FI_1 will be equal to 0 % since when one failure occurs, the duplicating agent is still available and it covers the same social knowledge. Fi_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. FI_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 [5] can be used to define equations that do not have restrictions on their usage.

Definition 4 (M-E graph)**.** 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 the following equations to compute minimal and maximal 1-failure impact:

$$Fi_1 = \frac{100}{N} \min_{i=1}^{M} \left\{ \sum_{a=1}^{N} \mathrm{COB}(a,i) \right\} [\%] \tag{1}$$

$$FI_1 = \frac{100}{N} \max_{i=1}^{M} \left\{ \sum_{a=1}^{N} \mathrm{COB}(a,i) \right\} [\%] \tag{2}$$

where $\mathrm{COB}(a,i)$ means that an end-agent a is "covered only by" a middle-agent i. This function is defined as follows:

$$\mathrm{COB}(a,i) = \begin{cases} 0, \exists k(\{v_k, w_a\} \in E(G) \wedge k \neq i), \\ 1, \text{otherwise.} \end{cases} \tag{3}$$

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:

$$Fi_2 = \frac{100}{N} \min_{i,j=1, i\neq j}^{i=M, j=M} \left\{ \sum_{a=1}^{N} COB(a,i,j) \right\} [\%] \tag{4}$$

$$FI_2 = \frac{100}{N} \max_{i,j=1, i\neq j}^{i=M, j=M} \left\{ \sum_{a=1}^{N} COB(a,i,j) \right\} [\%] \tag{5}$$

where $\mathrm{COB}(a,i,j)$ means that an end-agent a is "covered only by" middle-agents i and j. This function is defined as follows:

$$\mathrm{COB}(a,i,j) = \begin{cases} 0, \exists k(\{v_k, w_a\} \in E(G) \wedge k \neq i \wedge k \neq j), \\ 1, \text{otherwise.} \end{cases} \tag{6}$$

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 5. The minimal redundancy $\mathrm{MR}(G)$ of the graph G of type M-E is defined as follows:

$$\mathrm{MR}(G) = \min_{w \in V_e(G)} \{d_G(w)\}. \tag{7}$$

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 $\mathrm{MR}(G)$ should not be decreased. Assume that $E(H) = E(G) \cup \{e\}$. Then either $\mathrm{MR}(H) = \mathrm{MR}(G)$ or $\mathrm{MR}(H) = \mathrm{MR}(G) + 1$ holds, i.e., $\mathrm{MR}(H) \geq \mathrm{MR}(G)$. Thus MR is monotonic towards an edge addition and order-preserving.
- *Add a vertex to* $V_e(G)$ thus $\mathrm{MR}(G)$ should not be increased. Assume that $V(H) = V(G) \cup \{w\}$ where $w \in V_e(H)$. Then $\mathrm{MR}(H) = 0$, thus $\mathrm{MR}(H) \leq \mathrm{MR}(G)$ since $\mathrm{MR}(G) \geq 0$. Thus $\mathrm{MR}(G)$ is monotonic towards a vertex addition to $V_e(G)$ and order-reversing.
- *Add a vertex to* $V_m(G)$ thus $\mathrm{MR}(G)$ should not change. Assume that $V(H) = V(G) \cup \{v\}$ where $v \in V_m(H)$. Then $\mathrm{MR}(H) = \mathrm{MR}(G)$. Thus $\mathrm{MR}(G)$ 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[4] that can be used without specifying the number of simultaneous failures in which we are interested, we define *average redundancy*.

Definition 6. The average redundancy $\mathrm{AR}(G)$ of the graph G of type M-E is defined as follows:

$$\mathrm{AR}(G) = \frac{1}{N} \sum_{v \in V_m(G)} d_G(v) = \frac{1}{N} \sum_{w \in V_e(G)} d_G(w) = \frac{1}{N} |E(G)|. \tag{8}$$

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 $\mathrm{AR}(G)$ should not be increased. Assume that $E(H) = E(G) \cup \{e\}$. Then $\mathrm{AR}(H) = \mathrm{AR}(G) + \frac{1}{N}$, i.e., $\mathrm{AR}(H) > \mathrm{AR}(G)$. Thus AR is strictly increasing towards an edge addition.
- *Add a vertex to* $V_e(G)$ thus $\mathrm{AR}(G)$ should be decreased. Assume that $V(H) = V(G) \cup \{w\}$ where $w \in V_e(H)$. Then $\mathrm{AR}(H) = \mathrm{AR}(G)\frac{N}{(N+1)}$, i.e., $\mathrm{AR}(H) < \mathrm{AR}(G)$. Thus $\mathrm{AR}(G)$ is strictly decreasing towards a vertex addition to $V_e(G)$.
- *Add a vertex to* $V_m(G)$ thus $\mathrm{AR}(G)$ should not change. Assume that $V(H) = V(G) \cup \{v\}$ where $v \in V_m(H)$. Then $\mathrm{AR}(H) = \mathrm{AR}(G)$. Thus $\mathrm{AR}(G)$ 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.

[4] 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 σ-algebra a non-negative real number or ∞, 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 σ-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.

2.4 Social Knowledge Management in DHT Architecture

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 this section we describe breadth knowledge propagation, depth knowledge propagation, and no knowledge propagation. The efficiency of these three methods can be further improved by knowledge propagation on demand or by knowledge caching. To formally describe these methods, we first define neighbors, parents, and team members of a middle-agent.

Definition 7. Assume that a graph G is DHT with $G_1, \ldots, G_n$ its teams. Then we define all of the following:

1. $E^p(G) \subseteq E(G)$ as a set of edges where each $e \in E^p(G)$ represents a primary communication channel.
2. $\text{Neighbors}(v, G) = \{w | \{v, w\} \in E^p(G)\}$.
3. $\text{Parents}(v, G) = \{w | \{v, w\} \in E^p(G) \wedge \exists j \forall k (w \in V(G_j) \wedge v \in V(G_k) \rightarrow k > j)\}$.
4. If $v \in V(G_j)$ then $\text{TeamMememberss}(v, G) = V(G_j) \setminus \{v\}$.
   If $v \notin V(G_j)$ for every $j = 1, \ldots, n$ then $\text{TeamMememberss}(v, G) = \emptyset$.

2.4.1 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 is used for routing messages in the breadth knowledge propagation approach and holds for each middle-agent:

Definition 8 (full message routing)**.** 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 $v^{orig} \in V(G)$ or from an end-agent for which $v^{orig} \notin V(G)$. 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 $V^{orig}(m) \subset V(G)$ be a set of vertices stored in the message m such that $v \in V^{orig}(AddOrig(m, \{v\}))$[5]. 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 $KB(v)$ 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[6] m. Let $\text{Store}(v, w, c)$ be a subroutine that stores a reference to the

[5] 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 $V^{orig}(m')$ and avoid to contact these middle-agents, thus avoiding circuits in communication.

[6] Update subroutine is one of the main parts of a middle-agent where information from the message is processed and possibly stored to the internal knowledge base.

middle-agent w under the message 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:

IF Retrieve$(v, Context(m)) = \emptyset$ THEN
{ Update(v, m)
Let $R(v) = \{w | w \in Neighbors(v, G) \wedge w \neq v^{orig} \wedge w \notin V^{orig}(m)\}$
be a set of potential receivers.
Let $S(v) = \{w | w \in R(v) \wedge w \in TeamMemembers(v, G)\}$
FOR EACH $w \in R(v)$
{ IF $w \notin TeamMemembers(v, G)$ 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 types of 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 database 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. Since some of the communication channels can be faulty, the top-most team that consists of more than three middle-agents can have a loop of size greater than or equal to four. Therefore the context of the message is used to avoid these types of loops. 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 knowledge about all end-agents in the system is ultimately present in every middle-agent.

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 definition 4) 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 definitions 5 and 6) are computed as follows:

$$\mathrm{AR}(G) = \mathrm{MR}(G) = \frac{1}{N} \sum_{v \in V_m(G)} d_G(v) = \frac{1}{N} \sum_{v \in V_m(G)} N = \sum_{v \in V_m(G)} 1 = M \qquad (9)$$

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.

2.4.2 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 hold for each middle-agent:

Definition 9 (root message routing)**.** 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^{orig}:

IF Retrieve$(v, Context(m)) = \emptyset$ THEN
{ Update(v, m)
Let $R(v) = \{w | (w \in Parents(v, G) \cup TeamMemembers(v, G)) \wedge$
$\{v, w\} \in E^p(G) \wedge w \neq v^{orig} \wedge w \notin V^{orig}(m)\}$
Let $S(v) = \{w | w \in R(v) \wedge w \in TeamMemembers(v, G)\}$
FOR EACH $w \notin R(v)$
{ IF $w \notin TeamMemembers(v, G)$ THEN $Send(\{w\}, m)$
ELSE $Send(\{w\}, AddOrig(m, \{v\} \cup (S(v) \setminus \{w\})))$
}
IF $R(v) \neq \emptyset$ THEN $Store(v, v^{orig}, Context(m))$.
}

Definition 10 (parent retrieval message routing)**.** Apply the same set of assumptions as for the full message routing. Also let 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^{orig}:

IF Retrieve$(v, Context(m)) = \emptyset$ THEN
{ IF Process(v, m) THEN Send$(Retrieve(v, Context(m)), m)$
ELSE
{ FOR EACH $w \in Parents(v, G)$
Send$(\{w\}, m)$
IF Parents$(v, G) \neq \emptyset$ 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.

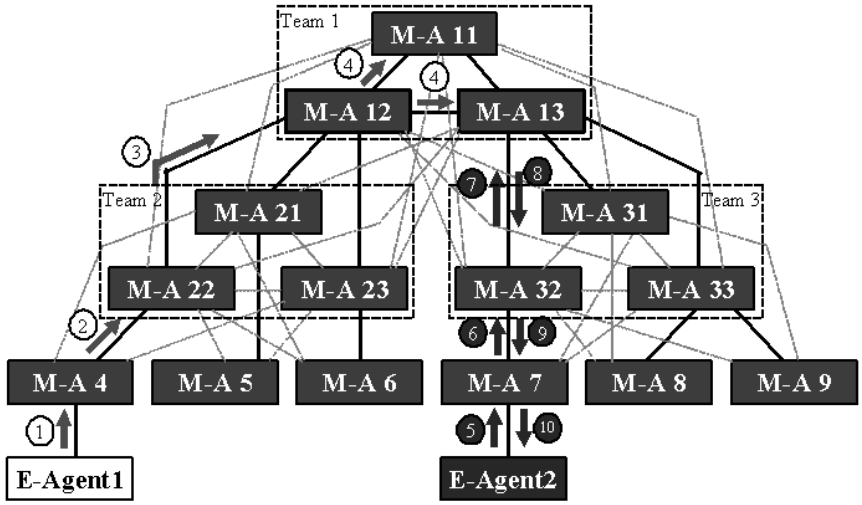
1. Registration, unregistration or modification types of messages are routed via the root message routing.
2. If a search request can be satisfied using local knowledge then reply with the result to the requester.
3. If a search request cannot be satisfied using local knowledge then it is routed via 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).

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 $AvHeight(G)$, where $v \in V(G)$ and $w \in V_1$ hold. 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 $MinHeight(G)$. Then the average redundancy and minimal redundancy are computed as follows:

$$\mathrm{AR}(H) = \frac{1}{N} \sum_{v \in V_e(H)} d_H(v) = AvHeight(G) + |V_1| \tag{10}$$

$$\mathrm{MR}(H) = \min_{v \in V_e(H)} \{d_H(v)\} = MinHeight(G) + |V_1| \tag{11}$$

Since the depth knowledge propagation uses the root message routing, social knowledge about an end-agent is on average propagated using $\mathrm{AvHeight}(G) + |V_1|$ messages (if we do not count possible acknowledgement messages). Social knowledge



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 to M-A 11 and 13.
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 it cannot be satisfied locally.
7. M-A 32 forwards the search request to M-A 13.
8. M-A 13 is able to satisfy the search request and replies with the search result to M-A 32.
9. M-A 32 propagates the search result back to M-A 7.
10. M-A 7 finally replies with the search result to E-Agent2.

Fig. 3. Depth knowledge propagation example. Only interactions that are directly related to the registration process and to the search process are described

is searched via the parent retrieval message routing; therefore, it uses on average $2AvHeight(G)$ messages.

2.4.3 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 [6] as JADE [8].

Definition 11 (full retrieval message routing)**.** Apply the same set of assumptions as for the parent retrieval message routing. Let 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 KB(v) and 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 KB(v). Let RetrieveExpectedReplies(v, c) be a subroutine that returns a set of vertices stored in KB(v) under the message context c. Let AddReply(v, c, m) be a subroutine that stores information from m to the database of v under the context c and GetReply(v, c) retrieves composite message based on previously stored information in 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^{orig}:

```
IF Retrieve(v, Context(m)) = ∅ THEN
{   IF Process(v, m) THEN Send(Retrieve(v, Context(m)), m)
    ELSE
    {   Let R(v) = {w | w ∈ Neighbors(v, G) ∧ w ≠ v^orig ∧ w ∉ V^orig(m)}
        Let S(v) = {w | w ∈ R(v) ∧ w ∈ TeamMememberss(v, G)}
        FOR EACH w ∈ R(v)
        {   StoreExpectedReply(v, w, Context(m))
            IF w ∉ TeamMemembers(v, G) THEN Send({w}, m)
            ELSE Send({w}, AddOrig(m, {v} ∪ (S(v) \ {w})))
        }
        IF R(v) ≠ ∅ THEN Store(v, v^orig, Context(m))
        ELSE Send({v^orig}, m)
    }
}
ELSE
    IF v^orig ∈ RetrieveExpectedReplies(v, Context(m)) THEN
    {   AddReply(v, Context(m), m)
        RemoveExpectedReply(v, v^orig, Context(m))
        IF RetrieveExpectedReplies(v, c) = ∅ THEN
            Send(Retrieve(v, Context(m)), GetReply(v, 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 types of 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^{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 or the requester can limit the number of results (for instance specified by FIPA).

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 when other methods are used. Assume a graph G of type M-E. The average redundancy and minimal redundancy are obviously equal to 1.

$$\mathrm{AR}(G) = \mathrm{MR}(G) = \frac{1}{N} \sum_{v \in V_e(G)} d_G(v) = \frac{1}{N} \sum_{v \in V_e(G)} 1 = 1 \tag{12}$$

Social knowledge about an end-agent is propagated using 1 message (if we do not 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.

2.4.4 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 the 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.

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 KQML specification [9].

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. The time stamping is well known mechanism used for instance for revocation of certificates [15].

2.4.5 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.

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 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.

2.5 Scalability of DHT Architecture

Although the term scalability is frequently used it is not precisely defined so far. Researchers in the parallel processing community have been using for instance Amdahl's Law and Gustafson's Law [7] and therefore tie notions of scalability to notions

of speedup. 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. Thus these definitions are not sufficient and we present several other ones.

"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." [16] "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." [12]

A formal definition of scalability in distributed applications is given in [19]. 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. 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.

To determine scalability without measuring resulting performance we can use definitions that are based on the extensibility and evaluate whether the cost to add new computing element is fixed. Thus we reformulate the scalability definition that is based on extensibility [12] to be used in the area of social knowledge architectures.

Definition 12 (fixed scalability)**.** Let $G = (V(G), E(G))$ be a graph that represents the structure of middle-agents and assume that G is of type[7] Θ. Then let H be a graph of type Θ 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 $|E^{\delta}(H)| \leq \varepsilon \cdot |V^{\delta}(H)|$ then G is called fixed scalable.

Theorem 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 λ 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[8]. Therefore $|E^{\delta}(H)| = \lambda$. We can repeat this process for all $v \in V^{\delta}(H')$ where $H' \supset G$, 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 $|E^{\delta}(H')| = \lambda \cdot |V^{\delta}(H')|$. □

Note that, for instance, the centralized architecture is obviously not fixed scalable since it cannot accommodate more than one middle-agent. Also, for instance,

[7] The type Θ is for instance DHT, structure defined by distributed matchmaking, teamwork-based, etc.

[8] Note that to ensure fixed scalability as defined above there is not requirement on the final structure of edges. Any structure that satisfies DHT type of graph is sufficient.

the teamwork-based technique [11] is not fixed scalable since the structure of middle-agents has to form a complete graph. Thus we present Table 1 of fixed scalability for various structures of social knowledge distribution.

Social knowledge distribution		**Fixed scalable?**
Centralized		No
Distributed	Acquaintance models	No†
Hybrid	Teamwork-based technique	No
	Distributed matchmaking	Yes
	Dynamic hierarchical teams	Yes

† 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 become fixed scalable.

Table 1. Fixed scalability of various types of social knowledge distributions

2.6 Reconfiguration in DHT Architecture

In Figure 1 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 in the case in which at least one of the following occurs:

- primary communication channels failed to transmit messages; or
- a receiving middle-agent failed.

These failures can be detected by various failure detection mechanisms, e.g., heartbeat mechanism [1], meta-agent observation [17], etc.

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

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 one. Note that the secondary communication channels replace only the primary ones 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.

3 EXPERIMENTS WITH DHT ARCHITECTURE

To test the robustness of the DHT architecture and to test various knowledge propagation methods in practical experiments, we created the test case setting as follows.

There are twelve possible types of end-agents. An end-agent that is registered into the system has one or two from the six possible capabilities. An end-agent location is randomly[9] chosen, i.e., an end-agent registers with one of the local middle-agents (M-As). The DHT structure of middle-agents has been practically tested in the Chilled Water System based on the Reduced Scale Advanced Development (RSAD) model that is a reconfigurable fluid system test platform, i.e., a chilled water part of a shipboard automation for US Navy vessels. Nevertheless, the following experimental results have been obtained on the same system where all M-As and end-agents run on a single computer Pentium III/800 MHz as separate threads under Agent-OS [14, 20]. 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:

   Create a new end-agent. A new end-agent of a randomly chosen type is created and registered to one randomly chosen M-A.

   Delete one of the existing end-agents. One randomly chosen end agent is unregistered from its local M-A and then the end-agent is deleted.

   Search for a capability. One randomly chosen end-agent sends a search request to its local M-A for a randomly chosen capability.

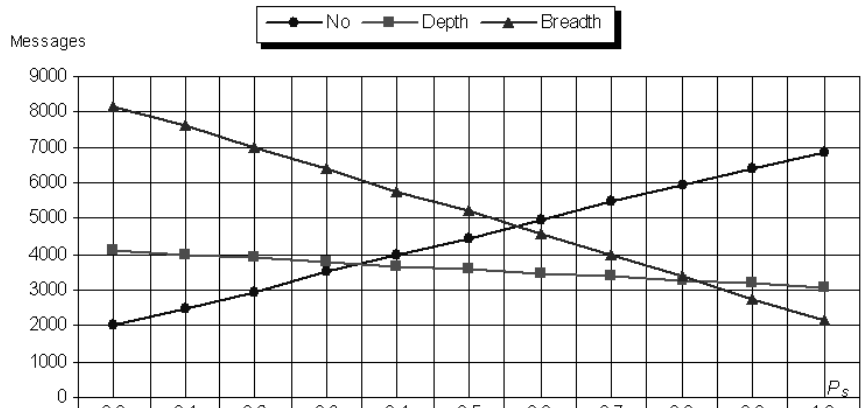
The distribution of probability to choose an action is as follows. The probability that the search action is chosen, denoted as P_S, is used as a parameter for each test case. The create and delete actions each has in every test case the same probability to be chosen, i.e., $\frac{1-P_S}{2}$.

3.1 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. The testing architecture consists of one global middle-agent (GM-A) and five local middle-agents that have GM-A as their parent. The goal of these tests is not to test robustness since there is only one GM-A used, but to compare presented knowledge propagation methods.

From the measurements presented in Figure 4 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 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 % and the breadth knowledge propagation method gives the best results otherwise. These results have been proved theoretically as well. The

[9] A uniform distribution of probability is used whenever we use the term randomly chosen.



	0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1,0
No	2040	2492	2967	3519	3971	4431	4955	5459	5955	6368	6827
Depth	4080	3978	3885	3778	3664	3596	3474	3381	3279	3175	3058
Breadth	8160	7621	7006	6387	5769	5188	4596	3967	3396	2759	2160

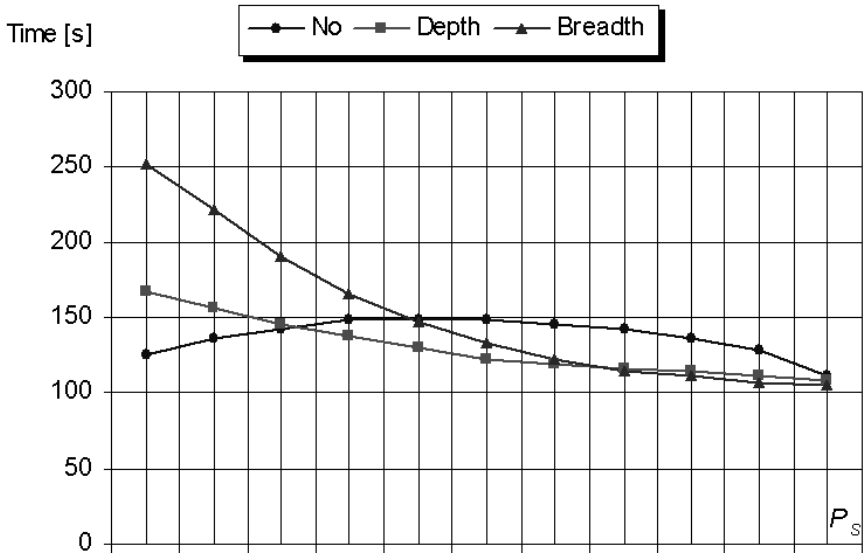
Fig. 4. Test case of knowledge propagation methods for the number of messages. The X-axis represents the probability that the search action is chosen, denoted as P_S. The Y-axis represents the total number of messages in the test run, where only messages that are related to the registration, unregistration, and to the search process are considered. Each value of each graph is the average (arithmetic mean) of 50 measurements

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.

3.2 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.

We can conclude from the measurements presented in Figure 5 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 4, 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



	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
No	125	136	143	148	148	148	145	142	136	129	111
Depth	168	157	146	137	130	123	119	116	114	112	108
Breadth	252	220	190	166	147	133	122	115	111	107	105

Fig. 5. The test case of knowledge propagation methods for the total running time. The X-axis 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

are significantly elevated since an agent has a very long creation time due to the full initialization of its planning templates.

3.3 Experiments with Robustness in DHT Architecture

To test the robustness of the DHT architecture in practical experiments, we created an experiment where the structure of middle-agents consists of three GM-As that form a team plus six M-As that are evenly distributed using their primary connections among the three GM-As. The test case should prove 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.

After 150 seconds the first GM-A simulates a failure and the system is not able to use it to propagate requests. The requests that are sent to this GM-A stay pending until the system discovers that it failed. The local M-As that are initially connected

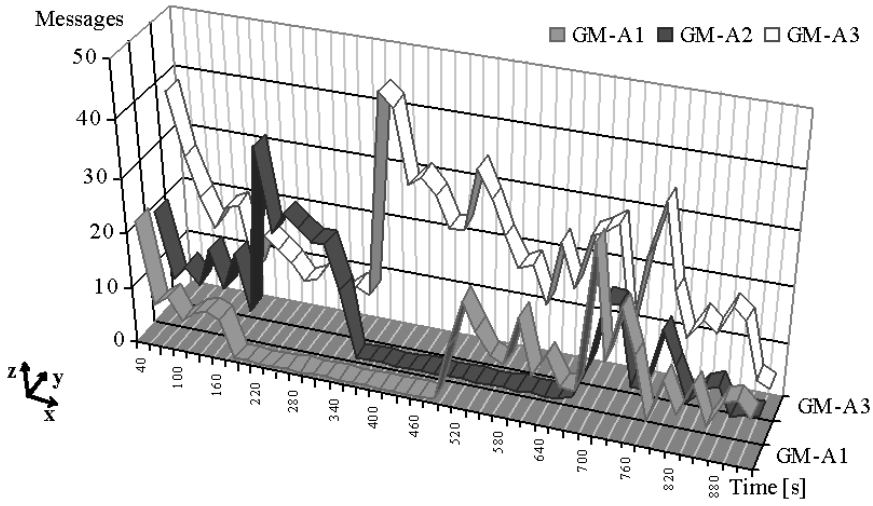


Fig. 6. Two failures of the global middle-agents in the DHT architecture with three members of the team. The architecture uses the depth knowledge propagation method. The graph depicts the communication frequency where the test run is split into time slots of 20 seconds (X-axis) with the number of messages 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, and so on

to the first GM-A dynamically switch to another GM-A, in this case to the second one.

After 300 seconds from the beginning of the test case also the second GM-A simulates a failure. In this case also the local M-As that are initially connected to the first and to the second GM-A dynamically switch to the third GM-A and the system is still able to respond to the incoming requests.

After 450 seconds from the beginning of the test case the first GM-A is repaired, followed by the second GM-A 150 seconds later. The local M-As dynamically switch back to their preferred GM-As again.

3.4 Comparison of Social Knowledge Distributions

To compare the robustness of various types of social knowledge distributions, we selected several examples of centralized, distributed, hierarchical, and DHT architectures. The goal is to evaluate these architectures based on the failure impact and on the average (AR) and minimal (MR) redundancy (see definitions 5 and 6), i.e., static impact of one or two simultaneous failures.

Social knowledge distribution		Fi_1 [%]	FI_1 [%]	Fi_2 [%]	FI_2 [%]	AR MR
Centralized	1 center	100	100	100	100	1
Distributed	2 centers (covering 40%,60%)	40	60	100	100	1
	N equally distr. ($N > 1$)	$\frac{100}{N}$	$\frac{100}{N}$	$\frac{200}{N}$	$\frac{200}{N}$	1
Hierarchical with 2 levels (DTH-1)	2 centers (40%,60%) + 1 glob.	0	0	0	60	2
	N equally distr. + 1 global	0	0	0	$\frac{100}{N}$	2
DHT with 1 team (2 levels)	2 centers (40%,60%)† + 2 glb.	0	0	0	0	3
	N equally distr. + 2 global†	0	0	0	0	3
	N global centers‡	0	0	0	0	N
Teamwork-based	N global centers	0	0	0	0	N

† 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.

Table 2. Failure impact and redundancy of various types of social knowledge distributions

From the results presented in Table 2 we can conclude that:

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

4 CONCLUSION

We have proposed and implemented the DHT structure of middle-agents that can be used to increase the fault-tolerance of a multi-agent system. We have 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, we have proved that the structure is maximally fault tolerant. We have proved that the DHT structure

is fixed scalable, i.e., can be scaled-up at a fixed incremental cost. We defined several methods for social knowledge propagation, such as breadth, depth, and no knowledge propagation and corresponding search techniques.

We have experimentally tested the proposed knowledge propagation methods in DHT structure to determine their advantages and disadvantages on a real multi-agent system. The experiments revealed that all proposed knowledge propagation methods can be efficiently used in presented testing environment based on the probability P_S that end-agents request the search action when measuring number of messages. We have experimentally tested robustness and reconfiguration of proposed architecture on a real multi-agent system with successful results.

REFERENCES

[1] Aguilera, M. K.—Chen, W.—Toueg, S.: Heartbeat: A Timeout-free Failure Detector for Quiescent Reliable Communication. In Proceedings of the 11th International Workshop on Distributed Algorithms, Springer-Verlag, Berlin, 1997, pp. 126–140.

[2] Byrne, C.—Edwards, P.: Refinement in Agent Groups. In (Weiss, G., Sen, S., eds.) Adaption and Learning in Multi-Agent Systems. Lecture Notes in Artificial Intelligence 1042, Springer-Verlag, Heidelberg, 1996, pp. 22–39.

[3] Chen, L.—Avizienis, A.: N-version Programming: A Fault-Tolerance Approach to Reliability of Software Operation. In Digest of Papers of the 8th Annual International Conference on Fault-Tolerant Computing, Toulouse, France, 1978.

[4] Decker, K.—Sycara, K.—Williamson, M.: Middle-Agents for the Internet. In Proceedings of the 15th IJCAI, Morgan Kaufmann, Nagoya, Japan, 1997, pp. 578–583.

[5] Diestel, R.: Graph Theory. Graduate Texts in Mathematics. Vol. 173, Springer-Verlag, New York, 2000.

[6] FIPA: The Foundation for Intelligent Physical Agents. `http://www.fipa.org`, Geneva, Switzerland, 1997.

[7] Gustafson, J. L.: Reevaluating Amdahl's Law. CACM, Vol. 31, 1988, No. 5, pp. 532–533.

[8] JADE: Java Agent DEvelopment Framework. Telecom Italia Lab, Torino, Italy, `http://sharon.cselt.it/projects/jade/`.

[9] Finin, T.—McKay, D.—Fritzson, R.: An Overview of KQML: A Knowledge Query and Manipulation Language, Technical Report, UMBC, Baltimore, 1992.

[10] Kraus, S.—Subrahmanian, V. S.—Tas, N. C.: Probabilistically Survivable MASs. In proceedings of IJCAI-03, Acapulco, Mexico, 2003, pp. 676–679.

[11] Kumar, S.—Cohen, P. R.: Towards a Fault-Tolerant Multi-Agent System Architecture. In Proceedings of the 4th International Conference on Autonomous Agents, Barcelona, Spain, 2000, pp. 459–466.

[12] Luke, E. A.: Defining and Measuring Scalability. In Scalable Parallel Libraries Conference, Mississippi, USA, 1994.

[13] Maříк, V.—Pěchouček, M.—Štěpánková, O.: Social Knowledge in Multi-Agent Systems. In Multi-Agent Systems and Applications, LNAI 2086, Springer, Berlin, 2001, pp. 211–245.

[14] Maturana, F.—Staron, R.—Tichý, P.—Šlechta, P.: Autonomous Agent Architecture for Industrial Distributed Control. 56th Meeting of the Society for Machinery Failure Prevention Technology, Sec. 1A, Virginia Beach, 2002, pp. 147–156.

[15] Naor, M.—Nissim, K.: Certificate Revocation and Certificate Update. In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, USA, 1998, pp. 217–228.

[16] Neuman, B. C.: Scale in Distributed Systems. In Readings in Distributed Computing Systems. IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 463–489.

[17] Pěchouček, M.—Macůrek, F.—Tichý, P.—Štěpánková, O.—Maříк, V.: Meta-agent: A Workflow Mediator in Multi-Agent Systems. In (Watson, I., Gordon, J., McIntosh, A., eds.) Intelligent Workflow and Process Management: The New Frontier for AI in Business IJCAI-99, Morgan Kaufmann Publishers, San Francisco, 1999, pp. 110–116.

[18] Pothipruk, P.—Lalitrojwong, P.: An Ontology-based Multi-agent System for Matchmaking. ICITA 2002, section 201-2, Bathurst, Australia, 2002.

[19] van Steen, M.—van der Zijden, S.—Sips, H. J.: Software Engineering for Scalable Distributed Applications. The 22nd COMPSAC '98, IEEE Computer Society 0-8186-8585-9, 1998, pp. 285–293.

[20] Tichý, P.—Šlechta, P.—Maturana, F.—Balasubramanian, S.: Industrial MAS for Planning and Control. In (Mařík, V., Štěpánková, O., Krautwurmová, H., Luck, M., eds.) Proceedings of Multi-Agent Systems and Applications II: 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001, LNAI 2322, Springer-Verlag, Berlin, 2002, pp. 280–295.

[21] Vadapalli, P.—Srimani, P. K.: A New Family of Cayley Graph Interconnection Networks of Constant Degree Four. In IEEE Transactions on Parallel and Distributed Systems, Vol. 7, 1996, No. 1, pp. 26–32.

Pavel Tichý graduated at the Czech Technical University in Prague (1997), assigned to Professor V. Mařík as a graduate student in the department of Cybernetics (PhD. – 2004). He specializes in fields of artificial intelligence and multi-agent systems. He has been working for the Rockwell Automation Research Center in Prague since 1995. His work includes automation, software development, enterprise controls, Shipboard Automation, and MAS tools and architectures.