

# Fault Tolerant and Fixed Scalable Structure of Middle-Agents

Pavel Tichý

Rockwell Automation Research Center & Czech Technical University,  
Prague, Czech Republic  
ptichy@ra.rockwell.com

**Abstract.** Middle-agents are used by end-agents to locate service providers in multi-agent systems. One central middle-agent represents a single point of failure and communication bottleneck in the system. Thus a structure of middle-agents can be 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  $\lambda$  has edge and vertex connectivity equal to  $\lambda$  and is maximally fault tolerant. We focus on social knowledge management describing several methods that can be used for social knowledge propagation and related methods to search for knowledge in this structure. We also test the fault-tolerance of this structure in practical experiments.

## 1. Introduction

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 [9]. 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 [1] 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 loses 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.

Several approaches have been used already to deal with these issues. Mainly the teamwork-based technique [7] 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 2.4. Another example is distributed matchmaking [13] that focuses on increasing the throughput 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 [9], but this technique implicitly does not ensure fixed scalability since in the worst case each agent keeps knowledge about all other agents.

## 2. Dynamic Hierarchical Teams Architecture

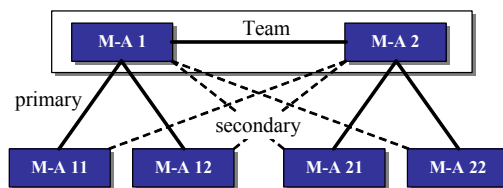
We propose the dynamic hierarchical teams (DHT) architecture to take advantage of both hierarchical and distributed architectures. The pure hierarchical architectures offer the scalability, but they are not designed to be fault-tolerant. On the other hand, the pure distributed architectures offer the 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 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 we call *teams* (see Fig. 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 2.5.

The DHT structure is not limited only to two levels, but it can support an  $N$ -level structure. For the cases where  $N > 2$  teams compose a hierarchical structure in

the form of a tree (see Fig. 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.

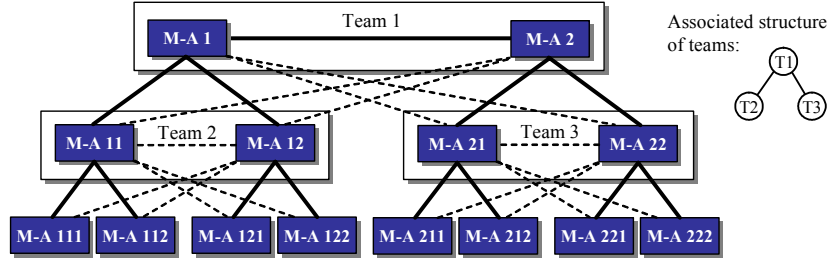


Fig. 2. Example of 3-level DHT architecture and associated structure of teams

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 [7]. 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 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** A graph  $G$  will be called DHT graph if there exist non-empty sets  $V_1, \dots, V_n \subset V(G)$  such that they are pairwise disjoint and  $V_1 \cup \dots \cup V_n \neq V(G)$ . 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)  $\forall v(v \in V(G) \setminus V_1 \rightarrow \exists j \forall w(w \in V_j \rightarrow \{v, w\} \in E(G)))$ <sup>1</sup>
- 2)  $\forall v(v \in V(G) \wedge v \notin V_1 \cup \dots \cup V_n) \rightarrow \exists! j \forall w(w \in V_j \rightarrow \{v, w\} \in E(G))$ <sup>2</sup>

<sup>1</sup> 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.

<sup>2</sup> 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  $v$ .

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

**Definition 2** The graph  $G$  is called DHT- $\lambda$  if  $G$  is DHT and  $|V_i| = \lambda$  for every  $i = 1, \dots, n$ , where  $\lambda \in \mathbb{N}$ .

## 2.2. Fault Tolerance in DHT Architecture

The fault tolerance of an undirected graph is measured by the vertex and edge connectivity of a graph [2]. 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  $G$ .

**Claim 1** If the graph  $G$  is DHT- $\lambda$  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- $\lambda$   $|V_j| = \lambda$ . Thus there are at least  $\lambda$  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 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 the resulting path in this case is  $vw_1w_2\dots w_k$ .  $\square$

**Lemma 1** If the graph  $G$  is DHT- $\lambda$  then  $G$  is  $\lambda$  vertex-connected and  $\lambda$  edge-connected.

*Proof.* 1) We prove that the graph  $G$  of type DHT- $\lambda$  is  $\lambda$  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  $v_2$  respectively) and ending at  $w_1$  (or  $w_2$  respectively) 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

<sup>3</sup> All members of each team except  $G_1$  are connected to all members of exactly one other team with lower index.

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- $\lambda$  is  $\lambda$  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  $v_2$  respectively) and ending at  $w_1$  (or  $w_2$  respectively) 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  $\lambda - 1$ .  $\square$

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

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

**Theorem 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 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 [2]. We already proved that  $\delta(G) = \lambda$  therefore  $\kappa(G) \leq \lambda$  and  $\lambda(G) \leq \lambda$ .  $\square$

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  $\delta(G)$  [16].

**Theorem 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  $\kappa(G) = \lambda$  and we also proved that it has the minimum degree  $\delta(G) = \lambda$ .  $\square$

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.

### 2.3. 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 3** Assume that a graph  $G$  is DHT with  $G_1, \dots, G_n$  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)  $\text{Neighbors}(v, G) = \{w \mid \{v, w\} \in E^p(G)\}$ .
- 3)  $\text{Parents}(v, G) = \{w \mid \{v, w\} \in E^p(G) \wedge \exists j \forall k (w \in V(G_j) \wedge v \in V(G_k) \rightarrow k > j)\}$ .
- 4) If  $v \in V(G_j)$  then  $\text{TeamMembers}(v, G) = V(G_j) \setminus \{v\}$ .  
If  $v \notin V(G_j)$  for every  $j = 1, \dots, n$  then  $\text{TeamMembers}(v, G) = \emptyset$ .

**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 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  $v \in V^{\text{orig}}(\text{AddOrig}(m, \{v\}))$ <sup>4</sup>. 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  $m$ <sup>5</sup>. Let  $\text{Store}(v, w, c)$  be a subroutine that stores a reference to the middle-agent  $w$  under the message context  $c$  into  $\text{KB}(v)$  and let  $\text{Retrieve}(v, c)$  be a subroutine that returns  $H \subseteq V(G)$  where a vertex  $w \in H$  iff  $\text{KB}(v)$  contains  $w$  under the message context  $c$ . Let  $\text{Context}(m)$  be a subroutine that returns context of a message  $m$ , i.e., the same value for all messages

<sup>4</sup> 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.

<sup>5</sup> 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.

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( $m$ ))  $\neq \emptyset$  THEN
{
  Update( $v$ ,  $m$ )
  Let  $R(v) = \{w \mid w \in \text{Neighbors}(v, G) \wedge w \neq v^{orig} \wedge w \notin V^{orig}(m)\}$  be a set of potential receivers.
  Let  $S(v) = \{w \mid w \in R(v) \wedge w \in \text{TeamMembers}(v, G)\}$ 
  FOR EACH  $w \in R(v)$ 
  {
    IF  $w \notin \text{TeamMembers}(v, G)$  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 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 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.

**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 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  $v^{orig}$ :

```

IF Retrieve( $v$ , Context( $m$ ))  $\neq \emptyset$  THEN
{
  Update( $v$ ,  $m$ )
  Let  $R(v) = \{w \mid (w \in \text{Parents}(v, G) \cup \text{TeamMembers}(v,$ 

```

```

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

```

**Definition 6** 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( $v, Context(m)$ )  $\neq \emptyset$  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).

**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 [3] as JADE [5].

**Definition 7** 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( $m$ ))  $\neq \emptyset$  THEN
{
  IF Process( $v$ ,  $m$ ) THEN Send(Retrieve( $v$ , Context( $m$ )),  $m$ )
  ELSE
    {
      Let  $R(v) = \{w \mid w \in Neighbors(v, G) \wedge w \neq v^{orig} \wedge w \notin V^{orig}(m)\}$ 
      Let  $S(v) = \{w \mid w \in R(v) \wedge w \in TeamMembers(v, G)\}$ 
      FOR EACH  $w \in R(v)$ 
      {
        StoreExpectedReply( $v$ ,  $w$ , Context( $m$ ))
        IF  $w \notin TeamMembers(v, G)$  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( $m$ ))
      ELSE Send( $\{v^{orig}\}$ ,  $m$ )
    }
}
ELSE
  IF  $v^{orig} \in RetrieveExpectedReplies(v, Context(m))$  THEN
  {
    AddReply( $v$ , Context( $m$ ),  $m$ )
    RemoveExpectedReply( $v$ ,  $v^{orig}$ , Context( $m$ ))
    IF RetrieveExpectedReplies( $v$ ,  $c$ ) =  $\emptyset$  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).

**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 [6].
- *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 [11].

**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.

#### 2.4. 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 [4] 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 definitions.

"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" [12].

"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" [8].

A formal definition of scalability in distributed applications is given in [14]. 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 [8] to be used in the area of social knowledge architectures.

**Definition 8** Let  $G = (V(G), E(G))$  be a graph that represents the structure of middle-agents and assume that  $G$  is of type<sup>6</sup>  $T$ . 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 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

---

<sup>6</sup> The type  $T$  is for instance DHT, structure defined by distributed matchmaking, teamwork-based, etc.

DHT. 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  $|E^\delta(H)| = \lambda \cdot |V^\delta(H)|$ .  $\square$

Note that, for instance, the centralized architecture is obviously not fixed scalable since it cannot accommodate more than one middle-agent. Also, for instance, the teamwork-based technique [7] 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.

**Table 1:** Fixed scalability of various types of social knowledge distributions

Social knowledge distribution		Fixed scalable?
Centralized		No
Distributed	Acquaintance models	No <sup>7</sup>
Hybrid	Teamwork-based technique	No
	Distributed matchmaking	Yes
	Dynamic hierarchical teams	Yes

## 2.5. Reconfiguration in DHT Architecture

In Fig. 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 (an observable failure).

The secondary communication channel does not mean ‘second’; there can be more than one secondary communication channel per 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.

<sup>7</sup> 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.

### 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<sup>8</sup> chosen, i.e., an end-agent registers with one of the local middle-agents (M-As). All M-As and end-agents run on a single computer Pentium III/800Mhz as separate threads under Agent-OS ([10] and [15]). 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 randomly chosen M-A.
  - b) *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.
  - c) *Search* for a capability. One randomly chosen end-agent sends a search request to its local M-A for a randomly chosen capability.

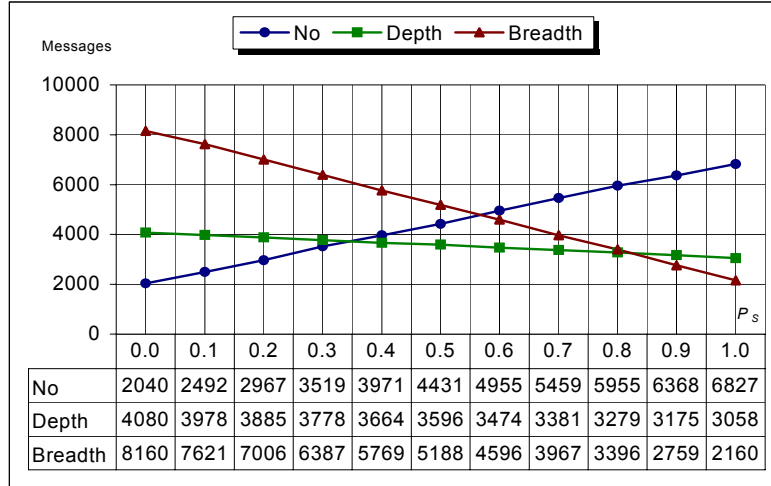
The distribution of probability to choose an action is as follows. The probability that the search action is chosen, denoted as  $P_S$ , is used as a *parameter* for each test case. The create and delete actions each has in every test case the same probability to be chosen, i.e.,  $(1-P_S)/2$ .

#### 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.

---

<sup>8</sup> A uniform distribution of probability is used whenever we use the term randomly chosen.

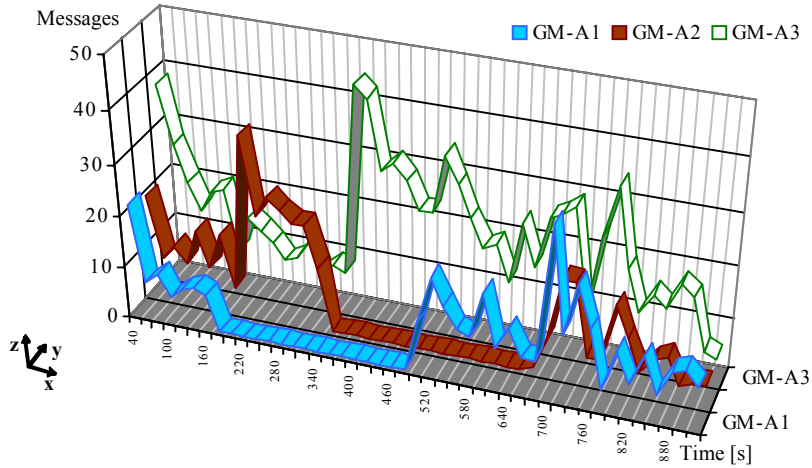


**Fig. 3.** Test case of knowledge propagation methods for the number of messages. The X-axis represents the probability that the search action is chosen, denoted as  $P_s$ . The Y-axis represents the total number of messages in the test run, where only messages that are related to the registration, unregistration, and to the search process are considered. Each value of each graph is the average (arithmetic mean) of 50 measurements.

From the measurements presented in Fig. 3 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 also theoretically. 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.

### 3.2. 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.



**Fig. 4.** 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.

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

#### 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 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. Decker, K., Sycara, K., Williamson, M.: Middle-Agents for the Internet. In Proceedings of the 15<sup>th</sup> IJCAI, Morgan Kaufmann, Nagoya, Japan (1997) 578-583
2. Diestel, R.: Graph Theory. Graduate Texts in Mathematics. Vol. 173, Springer-Verlag, New York (2000)
3. FIPA: The Foundation for Intelligent Physical Agents, <http://www.fipa.org>, Geneva, Switzerland (1997)
4. Gustafson, J.L.: Reevaluating Amdahl's Law. CACM, Vol. 31, No. 5 (1988) 532-533
5. JADE: Java Agent DEvelopment Framework, Telecom Italia Lab, Torino, Italy, <http://sharon.cselt.it/projects/jade/>.
6. Finin, T., McKay, D., Fritzson, R.: An Overview of KQML: A Knowledge Query and Manipulation Language, Technical Report, UMBC, Baltimore (1992)
7. Kumar, S., Cohen, P.R.: Towards a Fault-Tolerant Multi-Agent System Architecture. In Proceedings of the 4<sup>th</sup> International Conference on Autonomous Agents, Barcelona, Spain (2000) 459-466
8. Luke, E.A.: Defining and Measuring Scalability. In Scalable Parallel Libraries Conference, Mississippi, USA (1994)
9. Mařík, V., Pěchouček, M., Štěpánková, O.: Social Knowledge in Multi-Agent Systems. In Multi-Agent Systems and Applications, LNAI 2086, Springer, Berlin (2001) 211-245
10. Maturana, F., Staron, R., Tichý, P., Šlechta, P.: Autonomous Agent Architecture for Industrial Distributed Control. 56<sup>th</sup> Meeting of the Society for Machinery Failure Prevention Technology, Sec. 1A, Virginia Beach (2002) 147-156
11. Naor, M., Nissim, K.: Certificate Revocation and Certificate Update. In Proceedings of the 7<sup>th</sup> USENIX Security Symposium, San Antonio, Texas, USA (1998) 217-228
12. Neuman, B.C.: Scale in Distributed Systems. In Readings in Distributed Computing Systems. IEEE Computer Society Press, Los Alamitos, CA (1994) 463-489
13. Pothipruk, P., Lalitrojwong, P.: An Ontology-based Multi-agent System for Matchmaking. ICITA 2002, section 201-2, Bathurst, Australia (2002)
14. van Steen, M., van der Zijden, S., Sips, H.J.: Software Engineering for Scalable Distributed Applications. The 22<sup>nd</sup> COMPSAC '98, IEEE Computer Society 0-8186-8585-9 (1998) 285-293.
15. Tichý, P., Šlechta, P., Maturana, F., and 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: 9<sup>th</sup> ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001, LNAI 2322, Springer-Verlag, Berlin (2002) 280-295
16. 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, No. 1 (1996) 26-32