Grouping Algorithms for Scalable Self-Monitoring Distributed Systems

Benjamin Satzger and Theo Ungerer
Department of Computer Science
University of Augsburg
86159 Augsburg, Germany
{satzger, ungerer}@informatik.uni-augsburg.de

ABSTRACT

The growing complexity of distributed systems demands for new ways of control. Future systems should be able to adapt dynamically to the current conditions of their environment. They should be characterised by so-called self-x properties like self-configuring, self-healing, self-optimising, self-protecting, and context-aware. For the incorporation of such features typically monitoring components provide the necessary information about the system's state. In this paper we propose three algorithms which allow a distributed system to install monitoring relations among its components. This serves as a basis to build scalable distributed systems with self-x features and to achieve a self-monitoring capability. Evaluation measurements have been conducted to compare the proposed algorithms.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms

Keywords

grouping, failure detection, scalable, distributed system, algorithm, self-monitoring

1. INTRODUCTION

The initiatives Organic Computing (OC) [20] and Autonomic Computing (AC) [12, 13] both identify the exploding complexity as a major threat for future computer systems and postulate so-called self-x properties for these systems. To achieve these goals both the OC [15] and the AC community [13] regard monitoring information as a basis for organic or autonomic systems.

This paper proposes and compares three algorithms which establish monitoring relations within a distributed system. These algorithms are tailored to work very fast and autonomously in complex distributed environments. Based on the installed relations, monitoring services can be used for mutual monitoring of nodes. An instance for such a monitoring component is a failure detection service. Failure detectors generally provide information on failures of components of distributed systems. Typically distributed systems consisting of a finite set of processes or nodes are considered with a local failure detector attached to each process, see for example [4]. Failure detectors return a list of processes they are suspecting to have crashed. Obviously, within complex distributed systems, for scalability reasons it is not practicable that any two nodes are monitoring each other. Therefore strategies are needed to install monitoring relations among the components of a distributed system what can be seen as a self-monitoring capability.

The paper is organised in six sections. Section 2 gives a short overview of related work. In Section 3 the problem of establishing monitoring relations within a distributed system is stated. Then, Section 4 describes the proposed algorithms while Section 5 presents the evaluation results. Finally, Section 6 concludes the paper.

2. RELATED WORK

To supply adequate support for large scale systems, *hierarchical* failure detectors define some hierarchical organisation. Bertier et al. [3] introduce a hierarchy with two levels: a local and a global one, based on the underlying network topology. The local groups are LANs, bound together by a global group. Within each group any member monitors all other members. Different from Bertier et al. [3], in this work the existence of some classifying concept like a LAN is not required. Monitoring groups can be built also within a network of equal nodes. A hierarchy as proposed in [3] is not further investigated in this work, but could be easily built upon the *monitoring groups* which are introduced later on. Another hierarchical failure detector is presented by Felber et al. [7]. They emphasise the importance of well defined interfaces for failure detectors being able to e.g. reuse existing failure detectors.

Gossipping is a method of information dissemination within a distributed system by information exchange with randomly chosen communication partners. In 1972, Baker and Shostak [2] discussed a gossipping system with ladies and telephones. They investigated the problem of n ladies, each of them knows some item of gossip not known to the others. They use telephones to communicate, whereas the ladies tell everything they know at that time whenever one lady calls another. The problem statement was “How many calls are required before each lady knows everything?”. Demers et al. [6] pioneered gossipping in computer science as a way to update and ensure consistent replicas for distributed databases.

Van Renesse et al. [14] have been the first using gossipping for failure detection to cope with the problem of scalability. In their basic algorithm each process maintains a list with a heartbeat counter for each known process. At certain intervals every process increments its own counter and selects a random process to send its list to. Upon receipt of a gossip message the received list is merged with its own list. Each process also maintains the last time the heartbeat counter has increased for any node. If this counter is increased for a certain time then the process is considered to have failed. Additionally to this basic gossipping, the authors specify a multi-level gossipping algorithm that does not choose the communication partners completely randomly but dependent on the underlying network. Basically, they try to concentrate the traffic within subnets and to decrease it across them. Thus the scalability can be further improved. A disadvantage is that the size of gossip messages grows with the size of processes what causes a relatively high network traffic. Furthermore the timeout to prevent false detections has to be rather high and since every process checks failures of processes by its own, false detections cause inconsistent information.

The SWIM protocol, based on the work of Gupta et al. [10] and described in a paper of Das et al. [5], faces the mentioned drawbacks as it uses a separate failure detector and failure dissemination component. The failure detector component detects failures while the dissemination component distributes information about processes that have recently either left, or joined, or failed. Each process periodically sends a ping message to some randomly chosen process and waits for it to request. In this way failures can be detected and are then disseminated by a separate gossip protocol. The separation of failure detection and further components as proposed in [5] is taken up in this work. While the previous chapter introduces the failure detection component, here the dissemination component is proposed.

Horita et al. [11] present a scalable failure detector that creates dispersed monitoring relations among participating processes. Each process is intended to be monitored by a small number of other processes. In almost the same manner as in systems mentioned above, a separate failure detection and information propagation is used. Their protocol tries to maintain each process being monitored by k other processes. As a typical number for k they declare 4 or 5. When a process crashes, one of the monitoring processes will detect the failure and propagate this information across the whole system. In addition to the description of their failure detector, Horita et al. compare the overheads of different failure detection organisations in their paper. The grouping mechanism of Horita et al. [11] is based on a random construction of monitoring relations. Each node selects a certain amount of randomly chosen nodes which then serve as its surveillants. Hence, it is not taken into account how well a node is suited to monitor another. One motivation for this chapter is to take such an optimality criterion into account.

Graph partitioning represents a fundamental problem arising in many scientific and technical areas. In particular, understanding the graph as a network, it is a problem closely related to the problem approached in this work. Consider each partition of a network as a group of nodes which monitor each other. A *k-way partition* of a weighted graph is the partitioning of the node set into k disjoint subsets, so as to minimise the weight of edges connecting nodes in different partitions. This problem is known to be NP-hard [9] while many heuristics and approximation algorithms are known which aim at producing solutions close to the optimum. However, most of these techniques are not applicable to distributed environments and are therefore unsuitable to form monitoring groups.

An algorithm capable of solving a slightly modified k-way partition problem in a distributed way is presented by Roy et al. [16]. It is based on a stochastic automaton called influence model [1]. An influence model consists of a network of nodes which can take one of a finite number of statuses at discrete time steps. At each time steps the algorithm proposed in [16] performs the following: Each node picks a node as determining node with a certain probability and copies its status. By recursively performing these steps, partitions emerge. They argue that, under some constraints, their algorithm finds partitions which pass to the optimal partition with probability 1.

In this work two algorithms to partition a network into groups are introduced which is a problem very similar to graph partitioning. However, the problem investigated here is adapted to the needs of complex distributed systems.

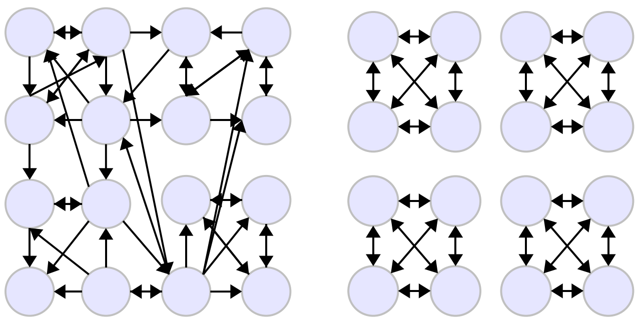
The contribution of this work is the introduction and evaluation of algorithms to form monitoring relations and monitoring groups respectively. The grouping component is independent from the used monitoring component. The latter could for instance be a failure detector as introduced in [17, 18] or any other mutual monitoring task. The separation of the monitoring itself and the group formation allows to create generic monitoring and grouping services. As clarified in the previous section, the separation of information propagation and monitoring has been identified as an important characteristic by many researchers. In the area of scalable failure detectors, the consideration of the suitability of monitoring relations has been neglected so far. For instance, the work of Horita et al. [11] proposes to choose surveillants *randomly*. Taking suitability information into account can improve the performance and reduce the overhead of monitoring components like failure detectors. Related methods from graph partitioning, which in fact search for optimal relations, are too complex and slow for an application in complex systems. Furthermore, graph partitioning algorithms normally need global knowledge and are not designed to work in a distributed environment. For reliable systems a fast installation of monitoring relations is more important than to find an optimal solution eventually. Especially from the point of view that a network can be subject to changes what means an optimal solution could become obsolete faster than finding it. To cover a wide range of different requirements and applications, dispersed monitoring relations, as arising if each node chooses its surveillants individually [11], as well as closed monitoring groups which result from e.g. network partitioning, are studied.

In the following, a precise formal definition of the stated problem is given.

3. PROBLEM STATEMENT

A *monitoring network* Net, a network of monitoring relations, is represented as a triple (N, M, s), where N is the set of nodes/processes of a network, $M \subseteq N \times N$ is the monitoring relation, and s is a function from $N \times N$ to a real value within $[0, 1]$. For each tuple $(u, v) \in N \times N$, $s(u, v)$ is the suitability of node u to monitor node v. This suitability can depend on different aspects like the latency of a connection, the reliability of a node, its load and so on. If a node u is not able to monitor another node v at all, $s(u, v)$ should output 0. The monitoring relation defines which monitoring relations are established, i.e. $(u, v) \in M$ means node u is currently monitoring node v. $(u, v) \in M$ is also denoted with $u \rightarrow v$. The relation M is irreflexive, i.e. it is not allowed that a node is monitoring itself. The term $\xrightarrow{*} v$ is defined as all nodes monitoring v, i.e. $\xrightarrow{*} v := \{u \in N \mid u \rightarrow v\}$. Similar, $u \xrightarrow{*}$ outputs all nodes u is monitoring, i.e. $u \xrightarrow{*} := \{v \in N \mid u \rightarrow v\}$.

The task of a grouping algorithm is basically, given a monitoring network $Net = (N, M, s)$ and a positive integer $m < |N|$, to establish monitoring relations such that every node of the network is monitored by at least m nodes. In this work two flavours of this problem are distinguished, namely *individual monitoring relations* also called *dispersed monitoring relations* and *closed monitoring groups*. In the former, monitoring relations can be set for each node individually while in the latter nodes form groups with mutual monitoring relations. The number m of surveillants for each node can be defined by the user. Typically, a higher number of surveillants provides a higher reliability but also causes a higher overhead. In Figure 1(a) an instance of individual monitoring relations of a monitoring network is illustrated with $m = 3$. Thereby, the illustration of the suitability information has been omitted. Figure 1(b) shows a corresponding partition of a network into monitoring groups.


(a) Individual monitoring relations

(b) Monitoring groups

Figure 1: Types of monitoring relations

In the following, problem definitions of establishing individual monitoring relations and monitoring groups are given.

3.1 Individual monitoring relations

Given a positive integer m, where $m < |N|$, establish monitoring relations M where $\forall n \in N$ holds $|\xrightarrow{*} n| = m$. This means each node is monitored by m other nodes. Furthermore, the algorithm should maximise the suitability of the grouping to establish adequate monitoring relations. Therefore the term

$$\sum_{v \in N} \sum_{u \in \xrightarrow{*} v} s(u, v)$$

should be maximised by the grouping algorithm. The optimisation of the suitability is a quality criterion for grouping algorithms, but it is not postulated that the algorithms output an optimal solution as it is more important to find solutions in all cases as fast as possible. The term *monitoring group* or simply *group* in the context of individual monitoring relations can be understood as all nodes monitoring one particular node, whereas the latter is the leader of the group. Thus, in a network of n nodes there are also n groups: each node $v \in N$ is the leader of the group $\{v\} \cup \xrightarrow{*} v$.

3.2 Closed monitoring groups

Different from the dispersed individual monitoring relations, a closed monitoring group is a group of nodes where all members monitor each other. This problem is very similar to a graph partitioning problem. In addition to the individual monitoring relations, constraints regarding the monitoring relations M are holding: M must be symmetric and transitive in order to produce closed monitoring groups. In another point, the problem of finding monitoring groups is relaxed, compared to individual monitoring relations, as it is not always possible to find groups of the size $m+1$ resulting in m surveillants per node in the group. If for instance a network has three nodes and monitoring groups of size 2 need to be established, this leads to an unsolvable problem. For such cases, also closed monitoring groups of bigger sizes are allowed. In detail, the problem $\forall n \in N$ holds $|\xrightarrow{*} n| = m$ is relaxed to $\forall n \in N$ holds $|\xrightarrow{*} n| \geq m$. A very simple solution to this problem is to combine the whole network into one group. This is a valid solution as just $|\xrightarrow{*} n| \geq m$ is postulated. However, the number of surveillants per node should be as close as possible to m. This represents a soft constraint similar to the maximisation of the suitability criterion.

Two nodes are in the same closed monitoring group if they are monitoring each other. An additional requirement for such monitoring groups is that each group has one node which is declared as leader. Such a role is needed by many possible applications based upon grouped nodes, e.g. to have one coordinator or contact for each group. An instance where one leader per group is necessary is the formation of hierarchical groups. Whether individual monitoring relations or monitoring groups are more adequate depends on the environment and the monitoring task. Furthermore, the installed groups can also be used for many other purposes beyond monitoring, like e.g. cooperative failure recovery. In [19] groups of nodes are formed which are planning together using an automated planning engine in order to recover the system. Such planning groups can also be established using the concepts introduced in this paper. Hence, many applications beyond monitoring are possible.

4. GROUPING ALGORITHMS

In this section three grouping algorithms are introduced, one to establish individual monitoring relations, two to form closed monitoring groups. The algorithms are tailored to solve these problems in a distributed manner. Furthermore, it is not assumed that all nodes have information about all other nodes what would simplify the problem significantly. The nodes of a self-monitoring network $Net = (N, M, s)$ do not know about the suitability s, i.e. how suitable other nodes are to monitor it, until they receive a message from a node with information about that. The suitability also might change over time. In the following the usage and relevance of suitability metrics for monitoring relations is discussed. Then, three algorithms are presented which provide the desired grouping capabilities. Being able to establish suitable monitoring relations the nodes of a network need information about each other. Such information might be the quality of the network connection of two nodes, the reliability of a node, and so on. Each node is holding relevant information about a number of other nodes allowing to compute suitability information.

The establishment of monitoring relations within a network $Net = (N, M, s)$ can be based on different aspects. Therefore the suitability function s has to be defined accordingly. Note that the suitability information typically is not computable before nodes receive information from other nodes. If it is for instance desired that nodes should be monitored by nodes with a similar hardware equipment and a fast network connection, the suitability function could be set to $s(u, v) = \frac{h(u,v)+n(u,v)}{2}$ where $h(u, v)$ returns a value within $[0, 1]$ indicating the similarity of the hardware equipment of u and v and $n(u, v)$ returns a value within $[0, 1]$ indicating the performance of the network connection. Such a scenario would make sense if a fast network connection improves the monitoring quality and in the case of an outage of a node, another node with similar hardware equipment is likely to have the ability to inherit the tasks of the failed node. Thus, the setting of the suitability function influences the establishment of monitoring relations. The definition of a suitability function should reflect the requirements of a monitoring system. All relevant factors should be included and weighted according to its importance.

Now three algorithms to establish monitoring relations in an autonomous distributed way are presented: Individual, which constructs individual monitoring relations, Merge and Species which install monitoring groups.

The idea of Individual is very simple: each node tries to identify the m most suitable nodes and asks them to monitor it.

In the initial state of the algorithm Merge, each node forms a group consisting of one node which it is leader of. Groups merge successively until they reach a size greater than m.

Species distinguishes between the two species *leader* and *non-leader*. The specificity of a node is random-driven. Non-leaders try to join a group whereas each group is controlled by one leader. In the case of an inadequate ratio of leaders to non-leaders, nodes can change its specificity.

4.1 Individual

Individual monitoring relations denote monitoring responsibilities set individually for each node. Using the suitability function, nodes can identify suitable surveillants. The most suitable ones are asked to monitor it. Therefore, nodes send monitoring requests to other nodes and wait for their acknowledgement. This process is repeated until the node has established m acknowledged monitoring relations. In Algorithm 1, the above described algorithm is formalised as pseudocode.

Algorithm 1 Individual

```
1: $id$                                    ▷ the id of this node
2: $m$                                     ▷ number of surveillants
3: $\mathcal{N}$                           ▷ set of known nodes
4: $id \xrightarrow{*} = \emptyset$        ▷ set of monitored nodes
5: $\xrightarrow{*} id = \emptyset$        ▷ set of surveillants
6:
7: loop
8:     if received message $msg$ from $n$ then
9:         if type of $msg$ is 'request' then
10:            $id \xrightarrow{*} = id \xrightarrow{*} \cup \{n\}$
11:            send('ack', $id$) to n
12:        end if
13:        if type of $msg$ is 'ack' then
14:            $\xrightarrow{*} id = \xrightarrow{*} id \cup \{n\}$
15:        end if
16:    else
17:        if $|\xrightarrow{*} id| < m$ then
18:            select most suitable node $n$ out of $\mathcal{N} \setminus \xrightarrow{*} id$
19:            send('request', $id$) to $n$
20:        end if
21:    end if
22: end loop
```

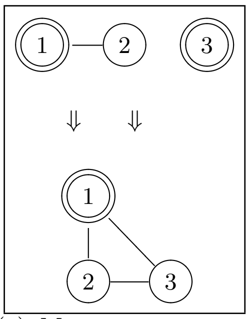
A further requirement for individual grouping algorithms which is omitted here could be that each node u monitoring a node v needs to know all other nodes also monitoring v, i.e. if $u \rightarrow v$ then u needs to know the set $\xrightarrow{*} v$. This might be necessary as in the case of a failure of v, all monitoring nodes could e.g. have to hold some kind of vote to gather a consistent view and to plan repairing actions respectively. This feature of closed monitoring groups could easily be integrated into Individual. This has not been done in order to investigate the more general algorithm as stated here.

4.2 Merge

In this section the Merge algorithm is discussed which establishes closed monitoring groups. Within these groups all nodes monitor each other. Every group has a group leader. Typically, the initial situation is a monitoring network $Net = (N, S, \varnothing)$ without monitoring relations and a number m which determines the desired number of surveillants. During the grouping of the nodes into monitoring groups, existing groups smaller than $m+1$ merge with other groups until the resulting group has enough members. Due to this mechanism the maximal size can be limited by $2 \cdot (m+1) - 1$. If there exists e.g. a group of size $2 \cdot (m+1)$ it can be splitted into two groups of valid size $m+1$. The group leaders which belong to a monitoring group smaller than $m + 1$, ask suitable other group leaders to merge their groups. If this request is accepted the groups merge whereas the requesting group leader must give off its leadership. The requested group leader is the leader of the newly formed group. After such a merging process the group leader informs all members about the new group. Nodes which lost the leadership adopt a completely passive role in the further grouping process and are not allowed to accept merging request from other leaders anymore.

Let us consider an example where m is 2, i.e. groups of minimum size 3 are formed. In Figure 2(a) two groups are examined, one consisting of Nodes 1 and 2 whereas Node

1 is leader and the other group consisting only of Node 3. Node 3 is requesting the group of Node 1 to merge. After the merge process a new group is formed with exactly 3 members and node 1 is leader of that group. Merge requests are never denied by leaders. Thus, as you can see in Figure 2(b), it is possible that groups emerge which have more than the desired $m + 1$ members.


(a) Merge process resulting in desired group size

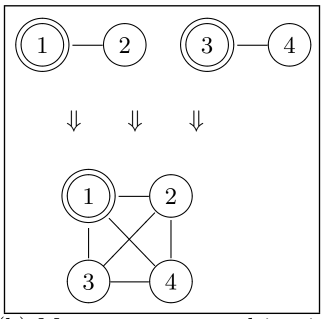

(b) Merge process resulting in oversized group

Figure 2: Merge scenarios

If groups become greater or equal to $2 \cdot (m + 1)$, as illustrated in Figure 3, a splitting is performed resulting in two monitoring groups which both have at least $m + 1$ members, what is enough to stop active merging activities. Thus the resulting group sizes of the MERGE algorithm are always between $m + 1$ and $2 \cdot (m + 1) - 1$.

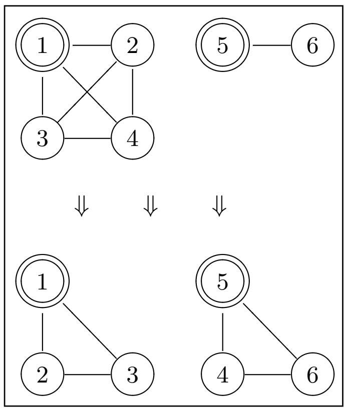


Figure 3: Merge and consecutive split

In Algorithm 2, the described grouping algorithm is formalised as pseudocode. Please note that only the most interesting parts of the algorithm are presented, due to space limitations. For instance, the notification of group members when the group has changed has been omitted.

4.3 Species

Like the MERGE algorithm, SPECIES also installs closed monitoring groups. It is based on the existence of two species: *leader* and *non-leader*. Leaders are group manager and each group contains exactly one leader. Non-leaders contact the most suitable leader trying to join its group. The specificity of a node is random-driven and dependent on the value of

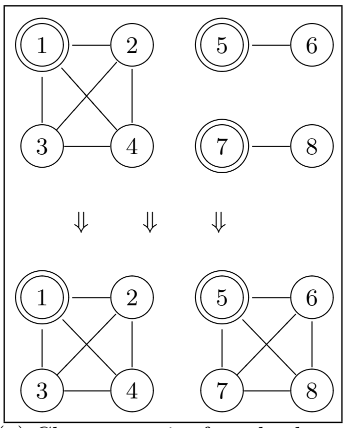
Algorithm 2 MERGE

```
1: $id$                                  ▷ the id of this node
2: $m$                                   ▷ minimum number of surveillants
3: $\mathcal{N}$                         ▷ set of known nodes
4: $\mathcal{G} = \{id\}$                ▷ set of group members
5: $l = id$                              ▷ leader, initially set to node id
6: $wr = F$                              ▷ is the node is waiting for a response
7:
8: loop
9:     if received message $msg$ from $n$ then
10:        if type of $msg$ is 'request' then
11:            if $id = l$ then
12:                if $wr$ then send ('waiting', $id$) to $n$
13:                else
14:                    if $|\mathcal{G}| + |msg.\mathcal{G}| \geq 2 \cdot (m+1)$ then
15:                        H = choose $\lfloor \frac{|\mathcal{G}|+|msg.\mathcal{G}|}{2} \rfloor - |msg.\mathcal{G}|$
16:                                group members to handover
17:                        send ('handover', $H$) to $n$
18:                    else
19:                        $\mathcal{G} = \mathcal{G} \cup msg.\mathcal{G}$
20:                        send ('ack',$\mathcal{G}$) to all $\mathcal{G} \setminus \{id\}$
21:                    end if
22:                end if
23:            else
24:                send ('non-leader', $\mathcal{G}$) to $n$
25:            end if
26:        else if type of $msg$ is 'ack' then
27:            $l = n$
28:            $\mathcal{G} = msg.\mathcal{G}$
29:            $wr = F$
30:        else if type of $msg$ is 'handover' then
31:            $\mathcal{G} = \mathcal{G} \cup msg.H$
32:            $wr = F$
33:        else if type of $msg$ is 'non-leader' then
34:            store information that $n$ is no leader
35:            $wr = F$
36:        else if type of $msg$ is 'waiting' then
37:            affects the selection of most suitable node
38:            $wr = F$
39:        end if
40:    else
41:        if $id = l \wedge |\mathcal{G}| < m + 1$ then
42:            select most suitable node $n$ out of $\mathcal{N} \setminus \overset{*}{\rightarrow} id$
43:            send('req', $\mathcal{G}$) to $n$
44:            $wr = T$
45:        end if
46:    end if
47: end loop
```

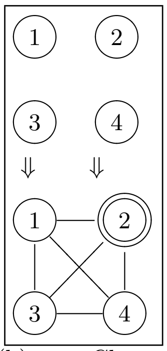
m. Consider a network consisting of n nodes. The optimal number of leaders is $\frac{n}{m+1}$ as the following example illustrates: Within a small network of 12 nodes, closed monitoring groups need to be installed with $m = 2$, i.e. two surveillants per node or groups of size three. The optimal case for that are four groups of size three. Thus $\frac{n}{m+1} = \frac{12}{3} = 4$ leaders are needed which the non-leaders can join. Therefore, the SPECIES algorithm selects every node as leader with probability $\frac{1}{m+1}$ and non-leaders otherwise. As it is worse to have too many leaders than too few, the probability of a node to become a leader can be adjusted to e.g. $\frac{0.8}{m+1}$. However, the random assignment of species to nodes does not guarantee a valid distribution into leaders and non-leaders. Thus, if a leader recognises that there are too many of them, they can toggle their species and transform into a non-leader. Vice versa, if nodes cannot find leaders to join they transform into a leader with a certain probability.

The network shown in Figure 4(a) contains too many leaders. In this case m is 3 which means groups of sizes of at least 4 need to be formed. However, this is not possible in this example. If no non-leader joins the groups smaller than 4, their leaders try to contact other leaders in order to find groups with enough members to poach some non-leaders. If this also fails, leaders then transform to non-leaders with a certain probability. This happens with Node 7 in this example. After that transformation a valid grouping is possible.

Figure 4(b) shows the contrary situation as above, where too few leaders are available, in this case even none. If non-leaders are unable to find any leader, they become leader with a certain probability.



(a) Change species from leader to non-leader



(b) Change species from non-leader to leader

Figure 4: Species scenarios

There are two cases how non-leaders join a group. If they do not belong to a group yet, they themselves care to find a group and join it. Leaders controlling an undersized group try to find oversized groups and ask their leaders to handover non needed members.

In Algorithm 3, the most important parts of SPECIES are formalised as pseudocode.

After the introduction of the proposed grouping algorithms, an evaluation is provided in the following section.

Algorithm 3 SPECIES

```
1: $id$                                  ▷ the id of this node
2: $m$                                   ▷ minimum number of surveillants
3: $\mathcal{N}$                         ▷ set of known nodes
4: $\mathcal{G} = \{id\}$                ▷ set of group members
5: $l = \begin{cases} id & \textit{with probability } \frac{0.8}{m+1} \\ undefined & \textit{otherwise} \end{cases}$
6: $wr = F$                              ▷ is the node waiting for a response
7:
8: loop
9:     if received message $msg$ from $n$ then
10:        if type of $msg$ is 'request' then
11:            if $id = l$ then
12:                $\mathcal{G} = \mathcal{G} \cup \{n\}$
13:            end if
14:        else if type of $msg$ is 'handover-request' then
15:            if $id = l$ then
16:                if $wr$ then
17:                    send ('waiting', $id$) to $n$
18:                else
19:                    x = $\min(\lfloor \frac{|\mathcal{G}|+|msg.\mathcal{G}|}{2} \rfloor - |msg.\mathcal{G}|,$
20:                    $|\mathcal{G}| - (m+1))$
21:                    H = choose x group members
22:                    to handover
23:                    send ('handover', $H$) to $n$
24:                end if
25:            else
26:                Forward message to random node
27:            end if
28:        else if type of $msg$ is 'handover' then
29:            $\mathcal{G} = \mathcal{G} \cup msg.H$
30:            $wr = F$
31:        else if type of $msg$ is 'chgspecies' then
32:            if $id = l$ then
33:                $\mathcal{G} = msg.\mathcal{G}$
34:            else
35:                Forward message to random node
36:            end if
37:        else if type of $msg$ is 'waiting' then
38:            affects the selection of most suitable node
39:            $wr = F$
40:        else
41:            if $id \neq l \land |\mathcal{G}| \leq 1$ then
42:                select most suitable node $n$ out of $\mathcal{N}$
43:                send('req', $\mathcal{G}$) to $n$
44:                $l = n$
45:            end if
46:            if $id = l \land |\mathcal{G}| < m + 1 \land$ with probability of 25% then
47:                if other suitable leader $n$ is known then
48:                    send('handover-request', $\mathcal{G}$) to $n$
49:                else if no leader can handover nodes then
50:                    change species to non-leader
51:                    $l$ = undefined
52:                    send('chgspecies', $\mathcal{G}$) to random node
53:                end if
54:            end if
55:            if $id = l \land$ no leader is known $\land$ with probability of 25% then
56:                change species to leader
57:                $l = id$
58:            end if
59:        end if
60:    end if
61: end loop
```

5. EVALUATION

In this section an evaluation for the above introduced algorithms is provided. For the purpose of evaluating and testing, a toolkit has been implemented which is able to simulate distributed algorithms based on message passing. It is written in JAVA and allows the construction of networks consisting basically of nodes, channels which connect two nodes, and algorithms running on nodes. As the simulation runs on one single computer, a random strategy selects the next node whose algorithm is executed partially. Thus, the asynchronous behaviour of distributed systems is covered. It is assumed that the communication channels do not drop messages and deliver them in the correct order.

The nodes of the monitoring network $Net = (N, M, s)$ used for the evaluation are theoretically arranged as a grid, as shown in Figure 5 for an example network consisting of 100 nodes. The nodes of the network are labelled with natural numbers which represent their ID. Note that the algorithms are neither based on that fact nor they take any advantage of that. The distance of two nodes u, v within the grid determines their mutual monitoring ability. Thus, the suitability has been set to the reciprocal value of the Euclidean distance of the nodes within the grid.

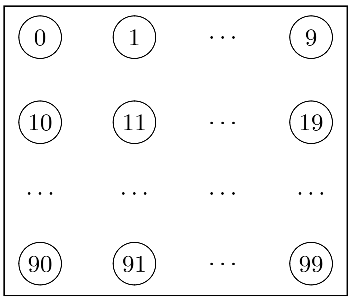


Figure 5: Evaluation network of 100 nodes

The evaluation network consists of 1000 nodes[1], where all nodes are able to communicate with each other. However, in most evaluation scenarios the nodes only have sufficient information about a certain number of nodes to compute a suitability value. This models the concept that in many networks nodes do not know everything but have a limited view.

The introduced grouping algorithms are evaluated within different scenarios. The evaluation focuses on the scalability of the establishment of monitoring relations, the optimality of the relations regarding the suitability metric, and the failure tolerance of a system if failure detectors are used together with the grouping approach. The evaluations have been conducted using different sets of parameters like the values for the desired number of surveillants m and the amount of information about other nodes. Each evaluation scenario has been replayed 1000 times whereas the results have been averaged.

Recall, a monitoring network Net is represented as (N, M, s), where N is the set of nodes of a network, $M \subseteq N \times N$ is the monitoring relation, and s is a function from $N \times N$ to a real value within $[0, 1]$. The task of a grouping algorithm is, given a positive integer $m < |N|$, to establish monitoring relations such that every node of the network is monitored by at least m nodes.

As suitability function $s(u, v)$, the reciprocal value of the Euclidean distance of the nodes u and v is used. At the beginning, the monitoring relation is empty, i.e. $M = \emptyset$. This means that the network is in a state where no monitoring relations are established yet. To model the fact that nodes usually do not have a complete view of the whole network, the value κ describes the part of the network each node is aware of. A value of $\kappa = 10$ means that each node has information about 10 randomly chosen nodes.

In the following the results of the conducted evaluations are presented.

5.1 Scalability

To establish monitoring relations, messages need to be sent. In the following this overhead is evaluated for the proposed grouping algorithms. All experiments have been conducted according to the description given above.

First the scalability regarding the network size is evaluated. In this experiment the number of desired surveillants m is set to 5 while each node knows 50 other nodes, i.e. $\kappa = 50$. Figure 6 shows the results of this experiment, whereas the values on the x-axis stand for the network size and the average number of messages sent by each node is depicted on the y-axis.

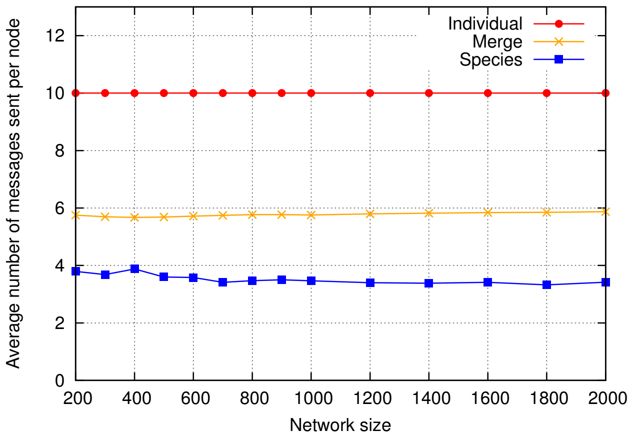


Figure 6: Scalability of grouping algorithms regarding network size ($\kappa = 50$)

As m is 5, each node executing the INDIVIDUAL algorithm needs 10 messages, 5 monitoring requests and 5 responses. MERGE needs less than 6 messages, SPECIES less than 4. The results indicate that all three algorithms can be classified as being independent from the network size, as the nodes basically do not send more messages within a bigger network. The algorithm SPECIES performs even better in bigger networks. The reason for this behaviour is the random-driven determination of the specificities. The aim of that process is to achieve a division into leaders and non-leaders of a defined ratio. In general, the bigger the network the better this ratio is met.

Thanks to the independence of the overhead caused by the

[1]Except for the measurements of the scalability regarding the network size where the number of nodes have been varied.

grouping algorithms from the network size, all introduced algorithms seem suitable to be applied within complex distributed systems.

All following evaluations are conducted with a network size of 1000 nodes.

To evaluate the overhead with regard to the sizes of the formed groups, the message sending behaviour of the algorithms is compared using different values for the minimum group size m within $[3, 4, \ldots, 20]$, whereas $\kappa = 100$.

Figures 7 shows the results of that experiment. It depicts the average number of sent messages on the y-axis. The x-axis stands for the different values of m.

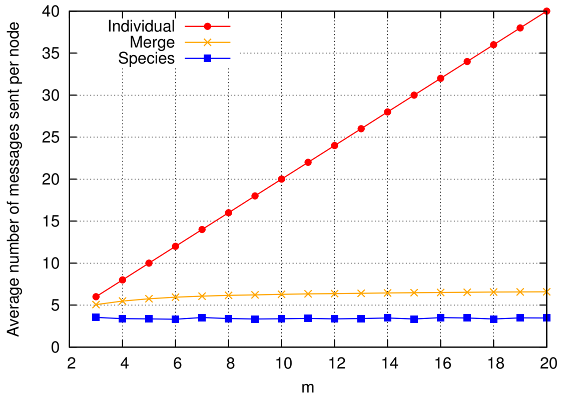


Figure 7: Scalability of grouping algorithms regarding group size ($\kappa = 100$)

The Species algorithm manages to group with the least messages of the three algorithms. The number of sent messages is totally independent from the number of surveillants. Individual scales linearly with the number of surveillants. The number of messages for Merge is strictly increasing, but it performs better than linear.

The three algorithms differ in the way they are able to meet the desired number of surveillants m. As it is mandatory to install at least m monitoring relations per node, only greater or equal values for the actually resulting group sizes are possible. Figure 8 shows the resulting number of surveillants in comparison to the value of m. Individual manages to installs exactly m surveillants per node. For closed monitoring groups it is a much harder problem to exactly meet this condition. Merge and Species typically form slightly larger groups in order to allow for a fast and robust grouping process.

The next section examines the monitoring relations with respect to their suitability according to the suitability function.

5.2 Suitability

As stated in the specification, the algorithms are supposed to take the suitability of the nodes into account. This means the term

$$\sum_{v \in N} \sum_{u \in \xrightarrow{*} v} s(u, v) \qquad (1)$$

should be maximised. The average suitability within the evaluation network is about 0.09. This means a random

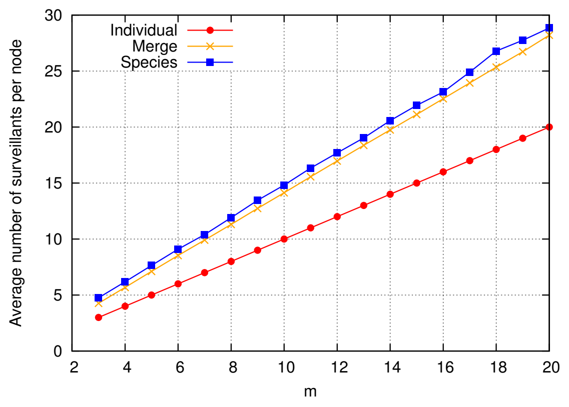


Figure 8: Resulting group sizes caused by different values for m

grouping produces monitoring relations of about that value.

Figures 9 and 11 show the results of the experiments concerning the suitability of the algorithms for different values of κ (10 and 100). This parameter represents the size of the nodes' view on the network. The x-axis represents the number of surveillants per node, the y-axis depicts the average suitability of the formed groups based on Equation 1.

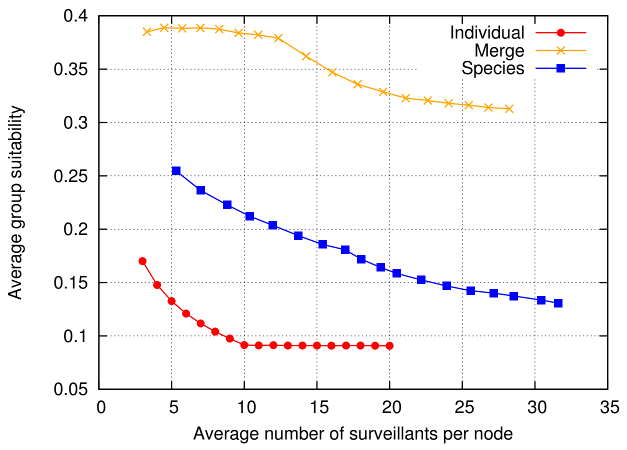


Figure 9: Suitability of grouping algorithms ($\kappa = 10$)

For all algorithms holds that bigger group sizes cause lower values for the suitability. Species and especially Merge handle grouping with limited information very well, while Individual performs optimal in the case of full information ($\kappa = 1000$) about the network.

5.3 Failure tolerance

In this section, the gain of applying the proposed grouping techniques with respect to failure tolerance is investigated. To evaluate the failure tolerance of the monitoring relations, the following methodology is used: It is assumed that a certain percentage of randomly chosen nodes within the network fail simultaneously, i.e. they crash and do not recover. Using failure detectors, nodes monitor each other according to the installed monitoring relations by a group-

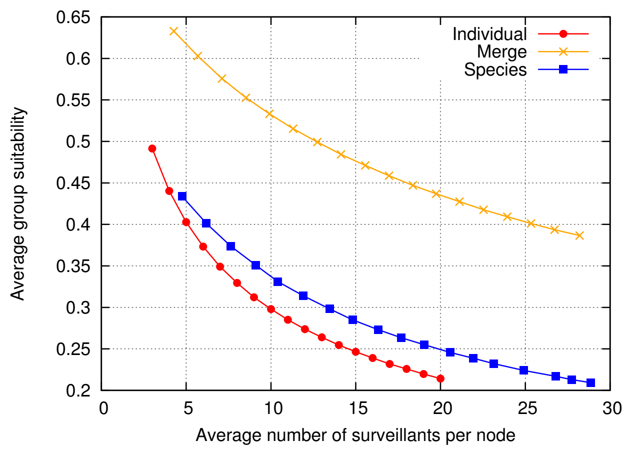


Figure 10: Suitability of grouping algorithms ($\kappa = 100$)

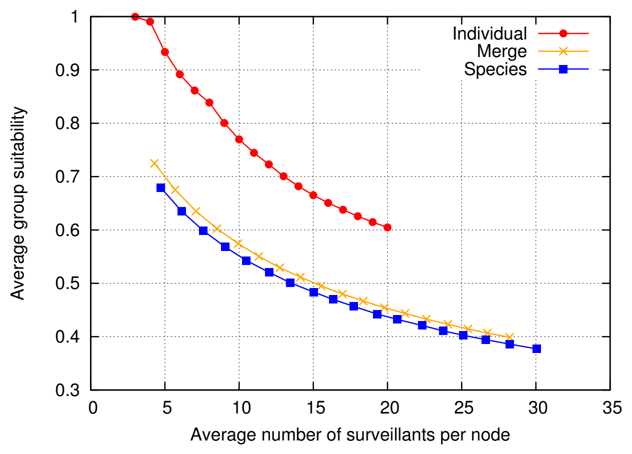


Figure 11: Suitability of grouping algorithms ($\kappa = 1000$)

ing algorithm. It is assumed that failure detectors eventually detect the failure of a node. An undetected failure means the failure of a node which remains undetected. In this setting this is only possible if a node and all its surveillants fail simultaneously.

The detection of a failure is the prerequisite of a subsequent repair or self-healing respectively. If a node has no surveillant, its failure equals an undetected failure. If in a network any node monitors all other nodes, only the complete failure of the whole network results in undetected failures. However, for more complex systems, the latter monitoring strategy typically introduces an excessive overhead. Before the evaluation results are presented, a short view on failure tolerance motivated by probability theory is given.

Let X be the number of elements within a set, $Y \leq X$ the number of elements within this set possessing a feature F, and $x \leq X$ the number of elements which are randomly chosen from the set. The probability of k elements with feature F being in the randomly chosen set is then

$$\frac{\binom{Y}{k}\binom{X-Y}{x-k}}{\binom{X}{x}},$$

according to the hypergeometric distribution [8].

Considering a network $Net = (N, M, s)$ and a number of surveillants per node of $m < |N|$. If ϕ random nodes of the network fail, where $m+1 \leq \phi \leq |N|$, the probability for the undetected failure of a certain node is

$$\frac{\binom{m+1}{m+1}\binom{|N|-m+1}{\phi-m+1}}{\binom{|N|}{\phi}} = \frac{\binom{|N|-m+1}{\phi-m+1}}{\binom{|N|}{\phi}}.$$

If ϕ is lower than $m+1$, the probability for an undetected failure is obviously 0. If for instance $\phi = 10\%$ of the nodes of a network $Net = (N, M, s)$ consisting of 100 nodes fail, whereas each node is monitored by $m = 3$ nodes, then the probability for a certain node $\eta \in N$ to fail undetectedly is:

$$\frac{\binom{|N|-m+1}{\phi-m+1}}{\binom{|N|}{\phi}} = \frac{\binom{100-4}{10-4}}{\binom{100}{10}} \approx 5 \cdot 10^{-5}.$$

The following simulations have been conducted as before with a network $Net = (N, M, s)$ of 1000 nodes. Monitoring relations are established with all three proposed grouping algorithms and different values for m. It is measured how many undetected failures occur if a certain percentage ϕ of random nodes fail.

Figure 12 presents the results for $\phi = 50\%$. The x-axis shows the average number of surveillants per node, the y-axis the number of undetected failures.

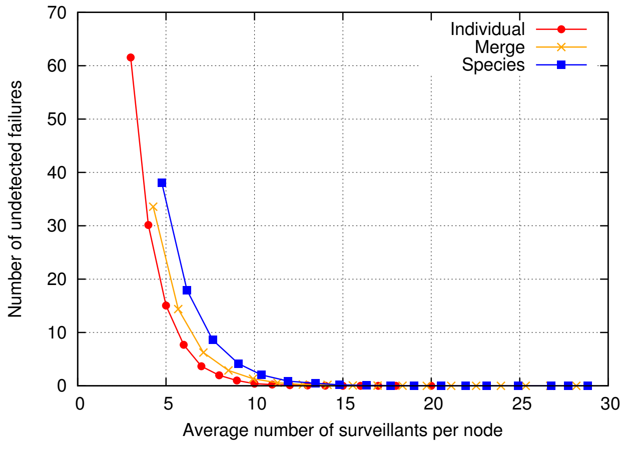


Figure 12: Failure tolerance of grouping algorithms (50% failure)

In all cases the number of undetected node failures decrease with a higher number of surveillants. As you can see, with a number of surveillants of 15, a failure of every second node in the network does not result in any undetected node failures. These results can be used as a utility to choose an adequate value for m, which is a balancing act between overhead and failure tolerance.

The algorithm INDIVIDUAL performs best because it has no variance in the number of surveillants. The parameter m exactly determines the resulting group size, i.e. the number of surveillants. For closed monitoring groups this number varies. The value for m only determines the minimum number of surveillants. Thus an average number of surveillants of e.g. 10 does not exclude groups of lower sizes. Consider for example a network of 10 nodes, arranged in two monitoring groups. In the first case two groups of sizes 5 (no variance), in the second case one group of size 4 and one group of size 6 (variance of 2). If for example only 4 nodes fail in the latter case undetected failures are possible, in the former not. The variance of the group sizes impacts the number of undetected failures, while a low variance is better. INDIVIDUAL has no variance, SPECIES has the highest variance, and MERGE's lies in between. The evaluation results reflect this fact.

6. CONCLUSIONS

In this work, the requirements for self-monitoring distributed systems are presented. The task is to autonomously install monitoring relations to enable self-monitoring distributed systems. The given formal problem statement is novel and takes suitability information into account. Three algorithms solving that problem are introduced and compared regarding their scalability, suitability, and the failure tolerance they are providing. The algorithms are tailored to install monitoring relations very fast what is important for reliable distributed systems.

7. REFERENCES

[1] C. Asavathiratham, S. Roy, B. Lesieutre, and G. Verghese. The influence model. *Control Systems Magazine, IEEE*, 21(6):52–64, Dec 2001.

[2] B. S. Baker and R. Shostak. Gossips and telephones. *Discrete Math.*, 2(3):191–193, June 1972.

[3] M. Bertier, O. Marin, and P. Sens. Performance analysis of a hierarchical failure detector. In *Proceedings 2003 International Conference on Dependable Systems and Networks (DSN 2003)*, pages 635–644, San Francisco, CA, USA, June 2003. IEEE Computer Society.

[4] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[5] A. Das, I. Gupta, and A. Motivala. SWIM: Scalable weakly-consistent infection-style process group membership protocol. In *DSN*, pages 303–312. IEEE Computer Society, 2002.

[6] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In *6th ACM Symposium on Principles of Distributed Computing*, pages 1–12, Vancouver, British Columbia, Canada, 10–12 Aug. 1987.

[7] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99)*, pages 132–141, Edinburgh, Scotland, 1999.

[8] W. Feller. *An Introduction to Probability Theory and its Application, Vol. 1.* John Wiley and Sons, New York, 1970.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[10] Gupta, Chandra, and Goldszmidt. On scalable and efficient distributed failure detectors. In *PODC: 20th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2001.

[11] Y. Horita, K. Taura, and T. Chikayama. A scalable and efficient self-organizing failure detector for grid applications. In *SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing CD*, pages 202–210, Seattle, Washington, USA, Nov. 2005. IEEE/ACM.

[12] P. Horn. Autonomic computing: Ibm's perspective on the state of information technology. *http://www.research.ibm.com/autonomic/*, 2001.

[13] J. O. Kephart. Research challenges of autonomic computing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 15–22, 2005.

[14] R. V. Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. Technical Report TR98-1687, Cornell University, Computer Science, May 28, 1998.

[15] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. In C. Hochberger and R. Liskowsky, editors, *INFORMATIK 2006 – Informatik für Menschen*, volume P-93 of *GI-Edition – Lecture Notes in Informatics*, pages 112–119, Bonn, Germany, Sept. 2006. Köllen Verlag.

[16] S. Roy, Y. Wan, and A. Saberi. *Algorithmic Aspects of Wireless Sensor Networks*, volume 4240/2006 of *LNCS*, chapter A Flexible Algorithm for Sensor Network Partitioning and Self-partitioning Problems, pages 152–163. Springer Berlin / Heidelberg, 2006.

[17] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer. A new adaptive accrual failure detector for dependable distributed systems. In *SAC '07: 22nd ACM symposium on Applied computing*, pages 551–555, New York, NY, USA, 2007. ACM.

[18] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer. Variations and evaluations of an adaptive accrual failure detector to enable self-healing properties in distributed systems. In *ARCS 2007, 20th International Conference, Zurich, Switzerland, March 12-15, 2007*, volume 4415 of *LNCS*, pages 171–184. Springer, 2007.

[19] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer. Using automated planning for trusted self-organising organic computing systems. In *5th International Conference on Autonomic and Trusted Computing (ATC 2008), pages 60-72, Oslo, Norway, June 23-25, 2008*, LNCS. Springer, 2008.

[20] H. Schmeck. Organic computing. *Künstliche Intelligenz*, 05(3):68–69, July 2005.