Towards trustworthy self-optimization for distributed systems

Benjamin Satzger, Florian Mutschelknaus, Faruk Bagci, Florian Kluge, and Theo Ungerer

Department of Computer Science
University of Augsburg, Germany
{satzger,bagci,kluge,ungerer}@informatik.uni-augsburg.de
http://www.informatik.uni-augsburg.de/sik

Abstract. The increasing complexity of computer-based technical systems requires new ways to control them. The initiatives *Organic Computing* and *Autonomic Computing* address exactly this issue. They demand future computer systems to adapt dynamically and autonomously to their environment and postulate so-called self-* properties. These are typically based on decentralized autonomous cooperation of the system's entities. Trust can be used as a means to enhance cooperation schemes taking into account trust facets such as reliability. The contributions of this paper are algorithms to manage and query trust information. It is shown how such information can be used to improve self-* algorithms. To quantify our approach evaluations have been conducted.

Key words: Trust, self-*, self-optimization, Organic Computing, Autonomic Computing

1 Introduction

The evolution of computer systems starting from mainframes towards ubiquitous distributed systems progressed rapidly. Common for early systems is the necessity for human administrators. Future systems, however, should act to a large extent autonomously in order to keep them manageable. The investigation of techniques to allow complex distributed systems to self-organize is of high importance. The initiatives Autonomic Computing [14, 5] and Organic Computing [3] propose systems with life-like properties and the ability to self-configure, self-optimize, self-heal, and self-protect.

Safety and security play a major role in information technology and especially in the area of ubiquitous computing. Nodes in such a system are restricted to a local view and typically no central instance can be responsible for control and organization of the whole network. *Trust* and *reputation* can serve as a means to build safe and secure distributed systems in a decentralized way. With appropriate trust mechanisms, nodes of a system have a clue about which nodes to cooperate with. This is very important to improve reliability and robustness in systems which depend on a cooperation of autonomous nodes.

In this paper we adopt the definition that trust is a peer's belief in another peer's trust facet. There are many facets of trust in computer systems. Such facets may concern for instance availability, reliability, functional correctness, and honesty. The related term *reputation* emphasizes that trust information is based on recommendation.

The development of trustworthy self-* systems concerns aspects of (1) generation of trust values based on direct experiences, (2) storage, management, retrieval of trust values, and (3) the usage of this information to enhance trustworthiness of the overall system. Generating direct trust values strongly depends on the trust facet. Direct experiences concerning the facet *availability* may be gathered by using heartbeat messages. The facet *functional correctness* of a sensor node may be estimated by comparison with measured values of sensors nearby. In this paper we will focus on (2) and (3), i.e. how to manage, access, and use trust information.

An instance of a distributed ubiquitous system which exploits self-* properties is our Smart Doorplate Project [11]. This project envisions the use of smart doorplates within an office building. The doorplates are amongst others able to display current situational information about the office owner and to direct visitors to his current location based on a location-tracking system. A middleware called "Organic Computing Middleware for Ubiquitous Environments" OCμ [10] serves as common platform for all included devices. The middleware system OCμ was developed to offer self-configuration, self-optimization, self-healing, and self-protection capabilities. It is based on the assumption that applications are composed of services, which are distributed to the nodes of the network. Service distribution is performed during the initial self-configuration phase considering available resources and requirements of the services. At runtime, resource consumption is monitored. In a former work we have developed a self-optimization mechanism [13, 12] to balance the resource consumption (load) between nodes by service transfers to other nodes. OCμ is an open system and designed to allow services and nodes from different manufacturers to interact.

In this work we incorporate a trust mechanism into our middleware to allow network entities to decide how far to cooperate with other nodes/services. This is used to enhance the self-optimization algorithm.

The paper is organized in seven sections. Section 2 gives an overview of the state of the art of trust in distributed systems. Section 3 presents the basic self-optimization algorithm we have developed. Section 4 introduces different algorithms to build a trust management layer which is able to provide functionalities covered by (2) as mentioned above. In Section 5 we present the trustworthy self-optimization, which extends the basic self-optimization and takes trust into account. This refers to (3). Then, Section 6 describes measurements of an implementation of the algorithm. Finally, Section 7 concludes the paper.

2 Related work

There are many approaches to incorporate trust into distributed systems. In this section some relevant papers are highlighted.

Repantis et al. [7] describe a middleware based on reputation. In their model, nodes can request data and services (objects) and may receive several responses from different nodes. In this case the object from the most trustworthy provider is chosen. The information about reputation of nodes is stored on its direct neighbors and appended to response messages. The nodes define thresholds for any object they request. Thus, only providers with a higher reputation are taken into account. After reception of an object the provider is being rated based on the satisfaction of the requester. In [7] nodes share one common reputation value which means that all nodes have the same trust in a certain node. In contrast, Theodorakopoulos et al. [9] describe a model based on a directed graph in which vertices are the network's nodes and the weighted edges describe trust relations. The weight of an edge (u, v) describes the trust of node u in v. Any weight consists of a trust value and the confidence in its correctness.

The TrustMe [8] protocol focuses on the anonymity of network members. It represents a technique to store and access trust information within a peer-to-peer network. The mining of trust information plays a minor role. An asymmetric encryption technique is used to allow for protection against attacks. In contrast to many trust management systems which support very limited anonymity or assume anonymity to be an undesired feature, TrustMe emphasizes the importance of anonymity. The protocol provides anonymity for the trust provider and the trust requester.

Cornelli et al. [4] present a common approach to request trust values: node A interested in the reputation of node B sends a broadcast message and receives response from all nodes which have a trust value about B. The latter message is encrypted by the public key of A. After reception of the encrypted answer the node contacts the responder to identify bogus messages.

In [6], a special approach is used to store trust information. It uses *Distributed Hash Tables* (DHTs) to store the trust value of a node in a number of parent nodes. These are identified by hash functions using the id of the child. A node requesting the trust value of a network member uses the hash function to calculate the parents which hold the value and sends a request to them.

Aberer et al. [2] present a reputation system to gather trust values. An interesting point is that trust values are mutual while traditionally nodes judge independently. The hope is to achieve an advantage due to the cooperation. This idea is integrated into the calculation of the global trust value of a node. The interaction triggered by the node itself as well as the requested interactions are accounted. Global trust values are binary, i.e. nodes are considered trustworthy or not. If nodes are cheating during an interaction they are considered globally untrustworthy. If a node detects a cheating communication partner it files a complaint. By the number of interactions with different nodes the probability rises that a liar is unmasked. In this model reputation values are stored within the network in a distributed way. This is done using the so-called PGrid [1].

3 Basic self-optimization algorithm

The basic self-optimization algorithm [13, 12] is inspired by the human hormone system. Its task is to balance the load of a distributed system based on services. This artificial hormone system consists of metrics which calculate a reaction (service transfer), nodes producing digital hormones which indicate their load, receptors collecting hormones and handing them over to the metrics, and finally the digital hormones holding load information. To minimize overhead the digital hormone value enfolds both, the activator as well as the inhibitor hormone. If the value of the digital hormone is above a given level, it activates while a lower value inhibits the reaction. To further reduce overhead, hormones are piggybacked to application messages and do not result in additional messages.

The basic idea behind the self-optimization is: When a heavy loaded node receives a message containing a hormone which states that the sender is lightly loaded, services are transferred to this sender. The metrics used to decide whether to balance the load between two nodes of the network are named transfer strategies because they decide on the transfer of a service.

Our self-optimization has the ability to improve a system during runtime. It yields very good results in load-balancing by only using local decision and with minimal overhead. However, it is not considered whether a service is transferred to a trustworthy node. Bogus nodes (e.g. nodes running a malicous middleware) might attract services in a systematic way and could induce a breakdown of the system. Unreliable faulty nodes might not have the ability to properly host services. The other way round, you would want to utilize particularly reliable trustworthy nodes for important services. Therefore, we propose to incorporate trust information into the transfer decision.

4 Trust management

As mentioned, a trust system needs a component which generates trust values based on direct experiences and it needs a component which is able to process this information. The generation of direct trust values depends strongly on the trust facet and the domain. Since a network can be seen as a graph, each node has a set of direct neighbors. In order to be able to estimate the trust facet *availability*, direct neighbors may periodically check their availability status. But depending on domain and application the mining of trust values differs. In a sensor network the measured data of a node can be compared with the measurements of its neighbors. In this way nonconforming sensors can be identified.

Since we do not focus on generation of trust values by observation but on the generic management of trust information we simply assume that any node has a trust value about its direct neighbors. This trust value $T(k_1, k_2)$ is within $[0, 1]$ and reflects the subjective trust of node k_1 in node k_2 based on its experiences. $T(k_1, k_2) = 0$ means k_1 does not trust k_2 at all while a value of 1 stands for 'full' trust. We assume that direct neighbors directly monitor each other to determine a trust value. This trust value might be inadequate due to insufficient monitoring

data. It may be possible that the trust is either too optimistic or too pessimistic. With continuous monitoring of neighbors their trust can probably be estimated better and better.

In the following three trust algorithms are presented which determine management of trust information in order to make it useful for the network's entities. The algorithms are explained being used together with an algorithm spreading information via hormones - like our self-optimization algorithm. However, the trust management is not limited to such a usage. It is assumed that nodes do not alter messages they are forwarding. In a real world application this should be assured by the usage of security techniques like encryption.

4.1 Forwarder

Only direct neighbors have the possibility to directly measure trust. Nodes that are not in the direct neighborhood need to rely on some kind of reputation. Also direct neighbors can use reputation as additional information. This first approach to propagate trust is quite simple. When a node B sends an application message to a node F, direct neighbor D which forwards the message appends its trust value $T(D, B)$ to it. Node F receives a message containing a hormone (with e.g. load information) and the trust of D in B as shown in Figure 1. This trust value is very subjective as only measured by one node, but this approach does not introduce additional messages for trust retrieval. Immediately after the receipt of an application message it has information about the trust of the sender.

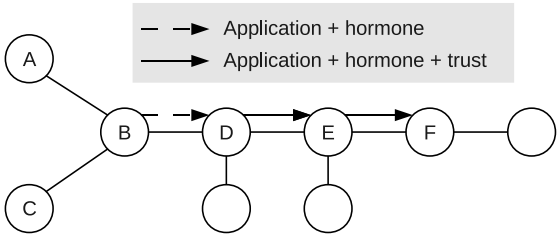

Fig. 1. Forwarder algorithm

4.2 Distant Neighborhood

In this approach not only the trust value of one direct neighbor is considered but all neighbors of the according node. A node sends an application message with an appended hormone to the receiver. If the receiver needs to know about the trust of the sender, it sends a trust request to all neighbors of the sender. They reply by a trust response message. Finally, trust in the sender is calculated as the average of all received trust values. This method results in much more

messages as the Forwarder algorithm would produce, but also provides more reliable information. A further advantage is the ability to detect diverged trust values. This might be used to identify bogus nodes. In Figure 2 node B sends an application message together with its capacity and load to node J which afterwards asks A, C, D, and E for their trust in node B.

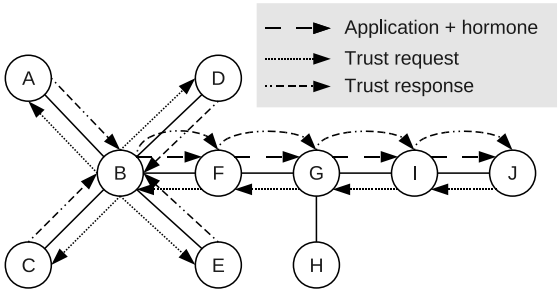

Fig. 2. Distant Neighborhood algorithm

4.3 Near Neighborhood

In this variant (see Fig. 3) a node spreads trust requests to query trust information, e.g. after the receipt of an application message. These trust requests are coupled with a hop counter. First, this hop counter is set to one which means that it first asks its own direct neighbors. If they have information about the target node they answer their trust value, otherwise they negate. If the node receives positive answers it averages the corresponding values. If the node receives only negative answers it increments its hop counter and repeats the trust request. At the latest when the request reaches a direct neighbor of the target node a trust value is responded. The requesting node stores the resulting trust value in order to answer other requests. Note that a node will execute this algorithm in order to update and refine its data even if it already has trust information about a target node. If a node already has a trust value of another node it is integrated into the averaging process. Initially, this algorithm produces much more messages than the ones described above. However, as trust values are distributed within the network the number of messages decreases over time because it becomes more and more likely that few hops are needed until a node is found with information about the target node. The same holds for the accuracy of trust values which increases with the runtime of the algorithm.

5 Trustworthy self-optimization

The self-optimization reallocates services during runtime to ensure a uniform distribution of load. The version described in Section 3 does not take the trust

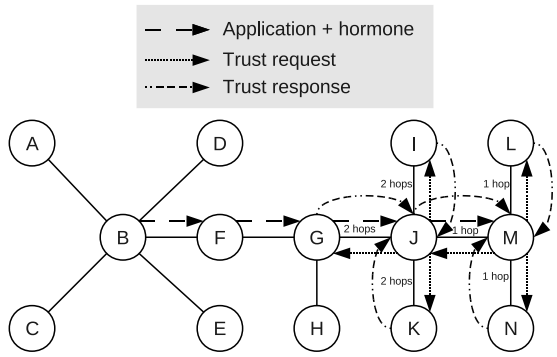

Fig. 3. Near Neighborhood algorithm

of nodes into account. In the following an approach is presented to incorporate trust into the self-optimization. Therefore, each of the three trust management algorithms presented above can be used. We assume that different services have different priorities. If a service is of high importance for the functionality of the system or collects sensitive data, its priority is high. The prioritization may be defined at design time or adapted dynamically during runtime. The trustworthy self-optimization aims at load-sharing with additional consideration of the services' priority, i.e. to avoid hosting of services with high priority on nodes with low trust.

A self-optimization step is performed strictly locally with no global control. Like the basic self-optimization algorithm, each node piggybacks hormones containing information about its current load and its capacity to each application message. This enables the receiver to compare this load value with its own load. Additionally the sender's trust can be queried via the proposed trust management layer. A transfer strategy takes this information and decides whether to transfer a service or not. If a service is identified it is tried to transfer it to the sender. This is only possible if the sender still has enough free resources to host this service. Due to the dynamics of the network this is not always sure.

The basic idea of the transfer strategy is to find a balance between pure load-balancing and trustworthiness of service distribution. Using parameter α allows to determine on which aspect to focus. A higher value for α emphasizes the need to transfer services to nodes with optimal trust values, a lower value for α results in a focus on pure load-balancing. All services B is able to host, for which A's trust into B is higher than their priority, and whose dislocation would balance the load significantly are considered for transfer.

6 Evaluation

Several test scenarios have been investigated in order to evaluate the trustworthy self-optimization. Each scenario consists of 100 nodes with random resource

capacity (e.g. RAM) and a random global but hidden trust value is assigned to each node. In real world applications the trust of a node must be measured. As this strongly depends on the trust facet and the application we have chosen a theoretical approach to simulate the trust by direct observation. It is based on the assumption that nodes are able to better estimate the trust of a node over time. In the simulation the trust of a node in its direct neighbor converges to the true hidden trust value with increasing number of mutual interactions as shown in Figure 4. In this example, the true trust value of the node is 0.5. Initially, the node is only able to estimate the trust very roughly while the error decreases statistically with the interactions. Formally, the trust of node r to node k after n interactions is modeled by

$$T_n(r,k) = t(k) + \rho_n.$$

In this formula $t(k)$ is the true global but hidden trust value of k and ρ_n is a random value which falsifies $t(k)$. With further interactions the possible range of ρ_i decreases, i.e. $|\rho_n| > |\rho_{n+1}|$. This random value simulates the inability to precisely estimate trustworthiness.

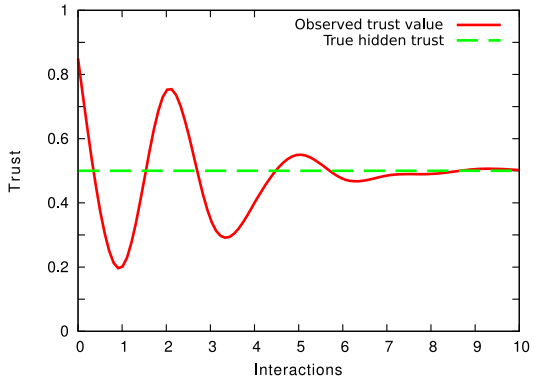

Fig. 4. Simulation of direct trust monitoring

In the simulation the nodes send dummy application messages to random nodes. These are used to piggy-back information necessary for self-optimization as described above. After reception of such a message the node determines the trust using a certain trust algorithm. Then, it is decided whether it is transferred or not. Initially, each node obtains a random number of services. Resource consumption and priority of a service are chosen randomly while the sum of all service weights is not allowed to exceed a node's capacity. The proposed trustworthy self-optimization is used for load-balancing and additionally tries to assign services with high priority to highly trusted nodes.

Rating functions are used to evaluate the fitness of a network configuration concerning trust and equal load-sharing. The main idea of the rating function for trusted service distribution f_T is to reward services with high priority and resource consumption running on very trustworthy nodes:

$$f_T = \sum_{n}^{N}(t(n) \sum_{s}^{S(n)} c(s) \cdot p(s)).$$

N is the set of all nodes, $S(n)$ is the set of services of a node n, $t(n)$ its true trust value, $c(s)$ and $p(s)$ resource consumption and priority of a service s.

The rating function for load-sharing f_L compares the current service distribution with the global theoretical optimal distribution. For each simulation the network consists of 100 nodes. At the beginning it is rated by f_T and f_L. Then the simulation is started and after each step the network is rated again. Within one step 100 application messages are randomly sent. This means that some nodes may send more than one message and others may send none to reflect the asynchronous character of the distributed system. The result of the receipt of an application message may be a service transfer dependent on the used trust algorithm and the node's load. Additionally, it is measured how many services are transferred. Each evaluation scenario has been tested 250 times with randomly generated networks and averaged values.

Figure 5 shows the gains of the trustworthiness of service distribution regarding the rating function f_T. Distant neighborhood reached the best results followed by Forwarder and Near Neighborhood. However, Forwarder introduces no additional message for trust value distribution. Without consideration of trust in self-optimization, services are transferred to random nodes and the overall trust is not improved, but even a little bit declined.

Figure 6 shows the network's load-sharing regarding the function f_L. Compared to the initial average load-distribution of 75% of the theoretical optimum, the self-optimization combined with any trust algorithm improves the load-balance. This means that the consideration of trust does not prevent the self-optimization to balance the load within the system. However, the quality of pure load-sharing cannot be reached by any trustworthy algorithm.

Distant Neighborhood performs best in the conducted measurements. It improves the trustworthiness of the the service distribution by about 20% while causing a deterioration of the load-sharing by about 4% (compared to load-sharing with no consideration of trust). This is supposed to be a beneficial trade-off as a slightly improved load-sharing will have weak impact on the whole system. However, running important services on unreliable or malicious nodes may result in very poor system performance. Forwarder avoids overhead due to queries, as trust values are appended to application messages. This algorithm improves the trustworthy service distribution by about 13% and decreases load-sharing by about 5%. Near neighborhood shows similar results as Forwarder, but its explicit trust queries cause a higher overhead. However, due to its working principle it may be the case that it would show better results after a longer experiment duration.

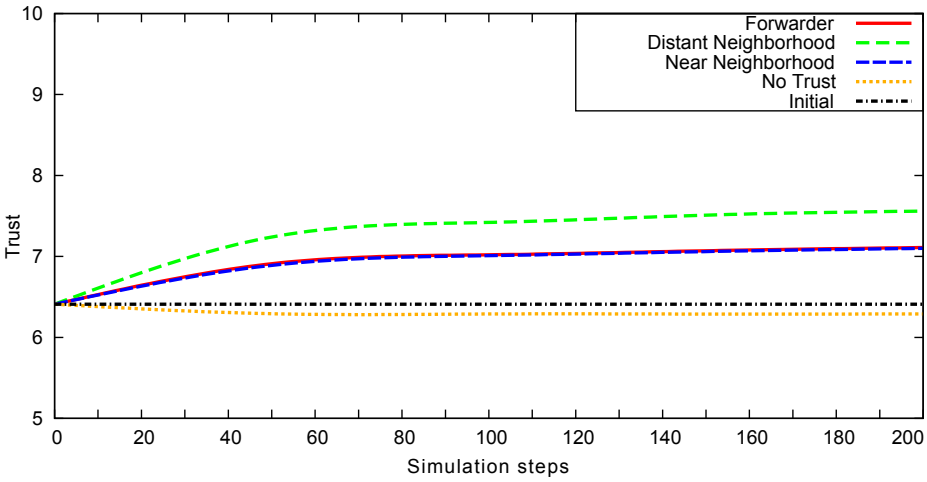

Fig. 5. Trustworthy service distribution (f_T)

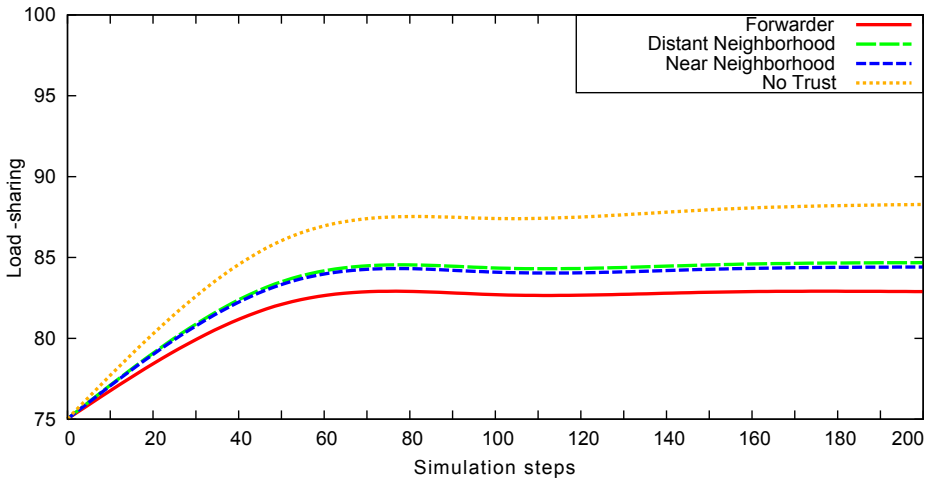

Fig. 6. Load-sharing (f_L)

7 Conclusion

This paper presents an approach for a trust management layer and shows how information provided by this layer can be used to improve a self-optimization algorithm. Three trust management algorithms have been introduced. With Forwarder, direct neighbors append its directly observed trust values automatically to each application message. Distant Neighborhood explicitly asks all direct neighbors of a node for a trust value. The last trust algorithm distributes trust values within the network for faster gathering of trust information especially after a longer runtime.

The trustworthy self-optimization does not only consider pure load-balancing but also takes into account to transfer services only to nodes regarded sufficiently trustworthy. A feature of the self-optimization is not to use explicitly sent messages but append information in the form of hormones to application messages to minimize overhead. Three different transfer strategies have been proposed which determine whether a service is transferred to another node or not regarding load and trust.

The presented techniques have been evaluated. The results show that trust aspects can be integrated into the system with little restrictions regarding load-balancing. The proposed trust mechanisms describe a way to increase the robustness of self-* systems with cooperating nodes.

References

1. K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Lecture Notes in Computer Science*, 2172, 2001.
2. K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *CIKM*, pages 310–317. ACM, 2001.
3. R. Allrutz, C. Cap, S. Eilers, D. Fey, H. Haase, C. Hochberger, W. Karl, B. Kolpatzik, J. Krebs, F. Langhammer, P. Lukowicz, E. Maehle, J. Maas, C. Müller-Schloer, R. Riedl, B. Schallenberger, V. Schanz, H. Schmeck, D. Schmid, W. Schröder-Preikschat, T. Ungerer, H.-O. Veiser, and L. Wolf. Organic Computing - Computer- und Systemarchitektur im Jahr 2010 (in German), 2003. VDE/ITG/GI position paper.
4. F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a P2P network. In *WWW*, pages 376–386, 2002.
5. P. Horn. Autonomic computing: IBM's perspective on the state of information technology. 2001.
6. S. D. Kamvar, M. T. Schlosser, and H. Garcia-molina. Eigenrep: Reputation management in p2p networks. In *in Proceedings of the 12th International World Wide Web Conference (WWW 2003*, 2003.
7. T. Repantis and V. Kalogeraki. Decentralized trust management for ad-hoc peer-to-peer networks. In S. Terzis, editor, *MPAC*, volume 182 of *ACM International Conference Proceeding Series*, page 6. ACM, 2006.
8. A. Singh and L. Liu. Trustme: Anonymous management of trust relationships in decentralized P2P systems. In N. Shahmehri, R. L. Graham, and G. Caronni, editors, *Peer-to-Peer Computing*, pages 142–149. IEEE Computer Society, 2003.

9. G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In *WiSe '04: Proceedings of the 3rd ACM workshop on Wireless security*, pages 1–10, New York, NY, USA, 2004. ACM.
10. W. Trumler. *Organic Ubiquitous Middleware*. PhD thesis, Universität Augsburg, July 2006.
11. W. Trumler, F. Bagci, J. Petzold, and T. Ungerer. Smart Doorplate. In *First International Conference on Appliance Design (1AD)*, pages 24–28, Bristol, GB, May 2003.
12. W. Trumler, A. Pietzowski, B. Satzger, and T. Ungerer. Adaptive self-optimization in distributed dynamic environments. In G. Di Marzo Serugendo, J.-P. Martin-Flatin, M. Jélasity, and F. Zambonelli, editors, *First IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pages 320–323, Cambridge, Boston, Massachussets, 2007. IEEE Computer Society.
13. W. Trumler, T. Thiemann, and T. Ungerer. An artificial hormone system for self-organization of networked nodes. In Y. Pan, F. J. Rammig, H. Schmeck, and M. Solar, editors, *Biologically Inspired Cooperative Computing*, pages 85–94, Santiago de Chile, August 2006. Springer-Verlag.
14. M. Weiser. The computer for the 21st century. 1995.