Improving the Self-X Properties of Organic Computing Systems with Trust

Nizar Msadek
Department of Computer Science
University of Augsburg
D-86159 Augsburg, Germany
Email: Nizar.Msadek@informatik.uni-augsburg.de

I. Introduction

The Organic Computing Initiative [1] has turned out to be a challenging vision for future information processing systems. This initiative consists of developing computer systems capable of so-called self-x properties (like self-configuration, self-optimization, self-healing and self-protection) to cope with the rapidly growing complexity of computing systems and to reduce the barriers that complexity poses to further growth. These properties are achieved by constantly observing themselves and initiating autonomous reconfiguration if necessary.

An essential aspect that becomes particulary prominent in this kind of systems is trust [2]. As part of my PhD thesis, a new design of self-x properties for organic computing systems will be investigated. Its main task is to improve self-x properties with trust capabilities to enable building a reliable system from unreliable components. The middleware system used in this work is the Trust-Enabling Middleware (TEM [3]) but these techniques can also be applied to any kind of distributed system.

This dissertation is part of the research unit OC-Trust of the German Research Foundation(DFG), which presented the following trust metrics to calculate the trust values required for the self-x properties. It is to note that all these trust metrics are integrated in TEM.

- **Direct Trust [4]** Is based on the experiences one has made directly with an interaction partner. Typically, trust values are calculated by taking the mean or weighted mean of past experiences.
- **Reputation [5]** Is based on the trust values of others that had experiences with the interaction partner. Reputation is typically raised if not enough or outdated experiences exist.
- **Confidence [6]** Before both values, direct trust and reputation, can be aggregated to a total trust value, the reliability of one's own trust value has to be determined, the so called confidence. If a node does have a direct trust value but is not confident about its accuracy, it needs to include reputation data as well.
- **Aggregation [7]** When all the aforementioned values are obtained, a total trust value based on the direct trust and reputation values can be calculated using confidence to weight both parts against each other. This value can then be used to improve the self-x properties.

The remainder of this paper is structured as follows. Section II introduces self-configuration, Section III gives an overview of self-optimization, Section IV describes mechanisms of self-healing and how to cope with failures. Section V presents the role of self-protection. Conflicting problems of trust values are discussed in Section VI. Finally, the scalability is shown in Section VII.

II. Self-Configuration

The approach of self-configuration is a crucial part for developing dependable and robust systems using self-x properties. This consists mainly of finding a robust distribution of services by including trust. The services are therefore categorized into important services i.e., with high trust level, and non important services i.e., with low trust level. The goal is to maximize the availability of important services. Therefore, it is necessary to assign the more important services to more reliable nodes. In addition to the reliability, resource requirements (e.g., like CPU and memory) should also be considered to be able to balance load of the nodes.

A. Metrics

The self-configuration focuses on assigning services with different trust levels to nodes such that the more important services are assigned to the more reliable nodes. Furthermore, the overall utilization of resources in the network should be well balanced. Therefore a metric is defined to calculate a Quality of Service (QoS_{total}).

$$QoS_{total} = (1 - \alpha) \cdot QoS_{trust} + \alpha \cdot QoS_{workload}.$$

The relationship between trust and workload can be set through α. α is constant $(0 \leq \alpha \leq 1)$ for a node. If $\alpha = 1$, the QoS_{total} is only obtained by the current value $QoS_{workload}$. If $\alpha = 0$, the QoS_{total} is decided only by the actual QoS_{trust} value. A higher value α favors $QoS_{workload}$ over QoS_{trust}.

- $\mathbf{QoS_{trust}}$ indicates how well the reliability of a node fulfilled the required reliability of a service. Figure 1 visualizes the possible situations that can occur in the calculation of the QoS_{trust}.

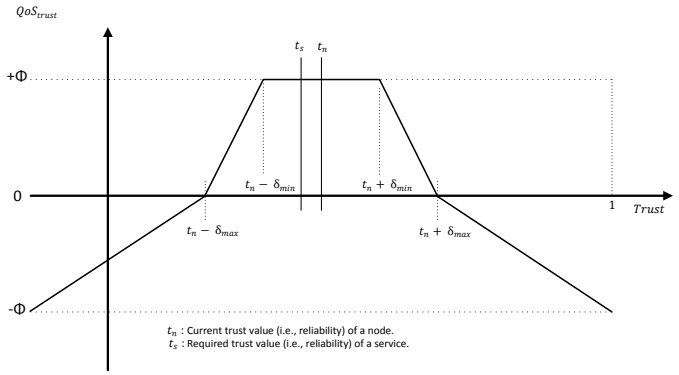

Fig. 1. Calculating QoS_{trust} based on the difference between the reliability of the node n (t_n) compared to the required reliability of the service s (t_s)

t_n represents the current trust value of a node n. t_s represents the required trust value of service s. If both of these values are close enough then n has fulfilled the required trust value of a service s. Close enough is defined by the threshold δ_{min}. If the difference between t_n and t_s is more than δ_{min}, then QoS_{trust} will be gradually decreased until it reaches $t_n \pm \delta_{max}$. If t_s is even beyond $t_n \pm \delta_{max}$ then the QoS_{total} will be fully decreased by ϕ where ϕ is the maximum value QoS_{trust} can decrease.

- $\mathbf{QoS_{workload}}$ is computed with the metric of Trumler [8]. As long as a node is not overburdened, the quality of service decreases linearly, otherwise it decreases exponentially.

B. Self-Configuration Process

This section discusses the methodology for distributing services. This consists of a collection of services with different priority levels which should run on nodes with different reliability levels. It is known to be a NP-hard problem to find an optimal solution for the distribution of the services on the nodes, such that the quality of service is optimal in terms of some predefined parameters [9]. Furthermore, there is no known polynomial algorithm which can, for a given solution, identify whether it is optimal. The aim behind self-configuration is to find a distributed and robust but not necessarily optimal, solution.

The quality of service metric presented in II-A is intended to evaluate the distribution phase which is based on the Contract Net Protocol [10]. During the distribution phase, every node on the network can act at different times or for different services as a manager or contractor. A *manager* is responsible for assigning services. A *contractor* is responsible for actual execution of the service. However, the manager is determined earlier by the user. Figure 2 visualizes a step-by-step example on how the negotiation process is run between nodes.

1) **Service Announcement:** The manager (e.g., node1) that has a service initiates contract negotiation by advertising the existence of that service to the other contractors (e.g., node2, node3 and node4) with a service announcement message. A service announcement can be transmitted to a single contractor in the network (unicast), to a specific set of contractors (multicast) or to all contractors (broadcast).
2) **Bidding:** Every contractor that receives the announcement calculates the $QoS_{workload}$ itself for the given service, i.e., based on its own locally available resources and then submit its bid in form of $QoS_{workload}$ to the manager back. Note that the service annoucement is ignored if the service cannot be hosted due to missing resources.
3) **Awarding:** If the expiration time has passed, the manager that sends the service announcement must calculate QoS_{trust} for every contractor in order to build the QoS_{total} and decides who to award the contract to. The result of this process will be then communicated to the contractors that submitted a bid. The expiration time is defined as a deadline for receiving bids. It is to note that the expiration time is determined earlier by the user.

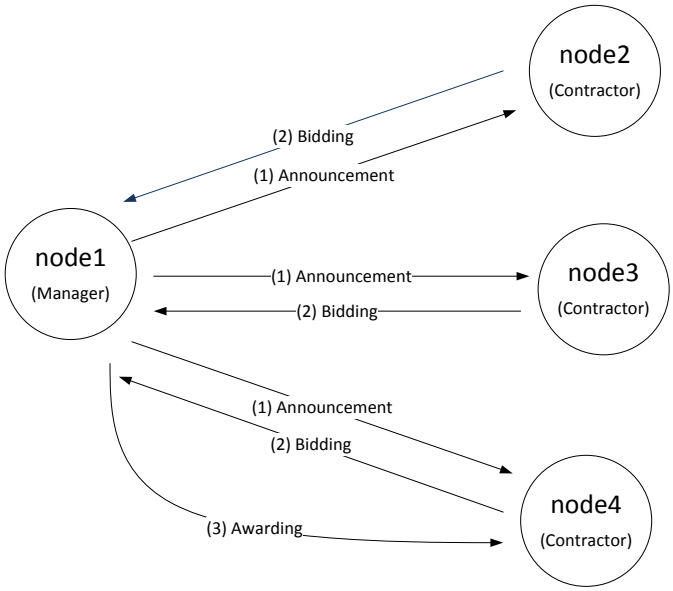

Fig. 2. Elementary representation of the distribution phase

C. Conflict Resolution

During the self-configuration process, several nodes could be ranked with the same QoS_{total}. This might lead to a conflict for the manager to decide to whom he awards the service. To avoid this a conflict resolution mechanism is used which does not need any further messages. The conflict resolution mechanism consists of three stages which might be used in the following chronological order:

1) **Minimum latency:** The node with the lowest latency will get the service.
2) **Minimum amount of already assigned services:** The node with the least amount of already assigned services will get the service, assuming that a lower amount of

services will produce less load (e.g., process or thread switching produces additional load).

3) **Node ID:** It is unlikely but not impossible that all of the former values were equal. In this case the id of the node will be used to find a solution to the conflict because every node has a unique id.

III. Self-Optimization

Based on the proposed self-configuration techniques, the services can be distributed on the nodes by different distribution strategies:

- **Uniform distribution:** The services are distributed on the nodes to evenly utilize all nodes and prevent single nodes to be overburdened. This leaves every node with a safety margin to cover possible performance spikes.
- **Power save distribution:** All services should run on a minimal number of nodes, so that free nodes can be deactivated in order to save energy.

Without trust, important services might run on unreliable nodes and are prone to failures. Such situations can be avoided. With trust, the reliability of a node can be measured and taken into consideration for the service distribution. For that reason, the distribution mechanisms should be investigated with and without a known reliability. The differences between using trust and not using trust have to be evaluated regarding the downtime of important services.

IV. Self-Healing

To investigate and research Self-Healing metrics, two ways have to be considered:

- **Proactive Self-Healing:** Enables to detect node instability prior to fail and then to move all running services by using self-configuration techniques to a more reliable node. False proactive shifts should be avoided.
- **Reactive Self-Healing:** Nodes save recovery information periodically during failure free execution. Upon failure, which has to be detected by using a failure detector, a node uses the already saved information to restart from an intermediate state i.e., called checkpoint, thus reducing the amount of lost computation.

V. Self-Protection

Trust values build the basis for all operations to increase the robustness of an organic computing system. Therefore, they must be specially protected against manipulation. Mármol and Pérez [11] presented some of the most important and critical security threat scenarios that can be found in the area of trust and reputation in a distributed system. Hence, all these scenarios have to be investigated and researched in order to make the self-x properties more resistent against such attacks.

VI. Problem of conflicting trust values

Another interesting point is to find a solution for conflicting trust values. This can happen by collecting reliability values independently from the neighbors of a node that can contradict each other. Figure 3 visualizes this problem in an example of three nodes.

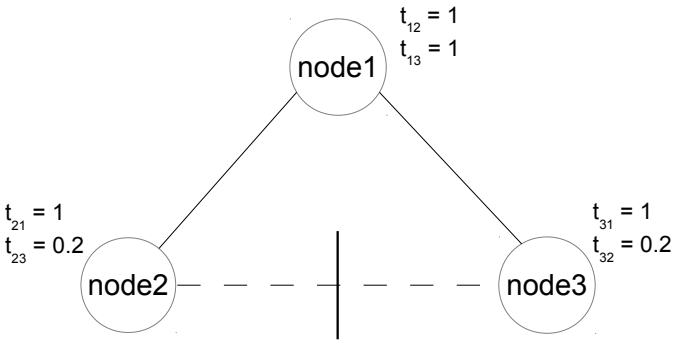

Fig. 3. Trust conflicting values in an example of three nodes

A shielding wall is set between two nodes i.e., node2 and node3 producing poor reliability values between these nodes, while a third node (node1) is not affected. t_{23} is the trust value node2 has about node3, so it wants to apply self-healing techniques in order to save all services running on node3, while node1 sees no need for action. Such situations must be omitted using metrics, which enable to deal with conflicting values.

VII. Dealing with large scale organic computing systems

In a hierarchical system, e.g., in a clustered Data Center, the already developed trust metrics are so far not entirely applicable, since only the next level in the hierachy is visible. Some nodes within a cluster could be less trustworthy. The cluster itself is still trusted because the cluster head is able to deal with its cluster members. Such situations can be omitted by using methods enabling the cluster head to control unreliable cluster members and to return a good result despite such members.

VIII. Related Work

The presented trust-enhanced self-x properties differ from state of the art selforganising mechanisms [12] [13] [1] [14] in three major points:

1) Development of techniques that allow the consideration of trust during analysis and interaction of Organic Computing systems.
2) Possibility to control complex systems with variable behavior.
3) Making a reliable and robust system out of unreliable components.

IX. Summary and Outlook

In this paper, a new design of self-x properties for organic computing systems is presented. Its main task is to improve using trust self-configuration, self-optimization, self-healing and self-protection. This approch will be embedded in a future work into the TEM, which is a trust enabling middleware implemented in Java and based on a peer to peer network.

Furthermore, two major existing problems in TEM are discussed, which are scalability and conflicting trust values.

ACKNOWLEDGMENT

This research is sponsored by the research unit OCTrust (FOR 1085) of the German Research Foundation (DFG).

REFERENCES

[1] C. Müller-Schloer, "Organic Computing - On the Feasibility of Controlled Emergence," *CODES + ISSS 2004. International Conference on Hardware/Software Codesign and System Synthesis, 2004.*, vol. 2-5, 2004.

[2] J.-P. Steghöfer, R. Kiefhaber, K. Leichtenstern, Y. Bernard, L. Klejnowski, W. Reif, T. Ungerer, E. André, J. Hähner, and C. Müller-Schloer, "Trustworthy Organic Computing Systems: Challenges and Perspectives ," *Proceedings of the 7th International Conference on Autonomic and Trusted Computing (ATC 2010), Springer*, vol. 14, pp. 62–76, 2010.

[3] G. Anders, F. Siefert, N. Msadek, R. Kiefhaber, O. Kosak, W. Reif, and T. Ungerer, "TEMAS - A Trust-Enabling Multi-Agent System for Open Environments," Universität Augsburg, Tech. Rep., 2013.

[4] R. Kiefhaber, B. Satzger, J. Schmitt, M. Roth, and T. Ungerer, "Trust measurement methods in organic computing systems by direct observation," in *IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC 2010)*, december 2010, pp. 105 –111.

[5] R. Kiefhaber, S. Hammer, B. Savs, J. Schmitt, M. Roth, F. Kluge, E. André, and T. Ungerer, "The neighbor-trust metric to measure reputation in organic computing systems," in *Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW 2011)*, october 2011, pp. 41 – 46.

[6] K. Rolf, A. Gerrit, S. Florian, U. Theo, and R. Wolfgang, "Confidence as a means to assess the accuracy of trust values," in *The sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO2012)*, september 2012.

[7] K. Rolf, J. Ralf, M. Nizar, and U. Theo, "Ranking of direct trust, confidence, and reputation in an abstract system with unreliable components," in *The seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO2013)*, september 2013.

[8] W. Trumler, "Organic ubiquitous middleware," Ph.D. dissertation, 2006.

[9] K. R. Reischuk, *Komplexitätstheorie: Band 1*. Teubner Verlag, 1999.

[10] R. G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver ," in *Defence Research Establishment Atlantic*. IEEE TRANSACTIONS ON COMPUTERS,, 1980, pp. 1–10.

[11] F. G. Mármol and G. M. Pérez, "Security threats scenarios in trust and reputation models for distributed systems," *Computers & Security*, vol. 28, no. 7, pp. 545–556, 2009.

[12] R. Urban, M. Moez, B. Jürgen, M.-S. Christian, and S. Hartmut, "Towards a generic observer/controller architecture for organic computing," *Bonner Köllen Verlag*, pp. 112–119, 2006.

[13] O. Jeffrey and C. David M, "The vision of autonomic computing," *IEEE Computer Society*, pp. 41 – 50, 2003.

[14] B. Jürgen, M. Moez, M.-S. Christian, P. Holger, R. Urban, R. Fabian, and S. Hartmut, "Organic computing addressing complexity by controlled self-organization," *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pp. 185 – 191, 2006.