Qualitative and Quantitative Aspects of Cooperation Mechanisms for Monitoring in Service-oriented Architectures

André Miede, Jean-Baptiste Behuet, Apostolos Papageorgiou, Michael Niemann, Nicolas Repp, Ralf Steinmetz, Fellow, IEEE

Multimedia Communications Lab (KOM), Technische Universität Darmstadt, Germany
Corresponding Author's E-Mail: Andre.Miede@KOM.tu-darmstadt.de

***Abstract*—Cooperation mechanisms for agents monitoring service-based workflows are a means to address the increasing complexity of modern enterprise architectures. These mechanisms are inspired by existing biological mechanisms and extend an existing decentralized monitoring architecture in order to handle deviations from Service Level Agreements autonomously. As the core cooperation mechanisms have been subject of our previous work and have been described in detail earlier, we present now an evaluation of our concepts regarding both effectiveness and efficiency, based on an implemented prototype.**

***Index Terms*—Agents, Cooperation, Self-organization, Service-oriented Architectures.**

I. Introduction and Scenario

Modern economies have become highly competitive and, therefore, require enterprises to adapt both quickly and continuously to changing circumstances and demands. The evolution of the underlying information systems (IS) and technologies is closely related to the businesses they support. Thus, IS face challenging requirements, such as high flexibility and adaptability [9]. While these requirements can be addressed properly in initial releases of IS, subsequent changes often decrease the system's adaptability seriously.

Maintaining the fulfilment of these requirements, i.e., using techniques such as loose coupling and interoperability can be achieved by the *Service-oriented Architectures* (SOA) paradigm [15]. SOAs are based on the "service" concept, where services can be seen as black boxes representing business functionalities and which are used to assemble business processes as service compositions. These processes and compositions may even cross enterprise boundaries, thus, enabling service-based, cross-organizational workflows [8][14] (cf. Fig. 1). In the last years, the SOA concept has become a successful way of addressing the enterprise issues mentioned above, e.g., using Web service technology as an implementation.

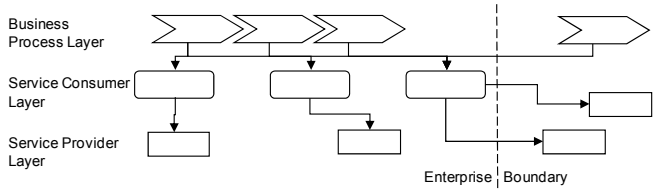

Fig.1 Service-based, cross-organizational workflow

However, as enterprise information systems increase in flexibility, they become more and more complex as well [9]. Complex, tedious management and maintenance tasks have to be performed manually by system administrators, leading to increased risk and costs. A possible solution to this problem is to introduce concepts of *self-organization* to the SOA domain. Self-organization is rooted in particular in the biological field, i.e., to successfully manage the complexity of living beings [13][18]. Inspired from biological concepts, the integration of self-organization mechanisms into SOAs is proposed as a way to improve their management and to reduce external human intervention [12].

The concrete application scenario for our research is the domain of cross-organizational workflows: in order to execute its business processes, an enterprise often needs to integrate third-party services offered by different external service providers. In our scenario, one enterprise plays the role of a service requester while its business partners are the service providers. Service clients and providers have to negotiate contracts defining the requirements for both parties. This way, Service Level Agreements (SLAs) are contracted, i.e., specifying service performance and availability by metrics such as response time and throughput [10]. However, the specification of the SLAs itself is no guarantee. The monitoring of their fulfilment during runtime is needed as well, as the actual service performance and availability have to be compared with the contracted ones. In addition, potential deviations from the defined requirements have to be handled timely and effectively.

The goal of this paper is to evaluate self-organization mechanisms designed and implemented for the SOA paradigm, i.e., cooperation mechanisms between agents for SLA monitoring and deviation handling.

The rest of this paper is structured as follows. Section 2 gives an overview of related work. The successive Section 3 presents the basic ideas of our cooperation concepts and describes important implementation details. Thereafter, Section 4 discusses the evaluation methods and the obtained qualitative and quantitative results. The paper closes with conclusions and an outlook on future research.

II. Related Work

The related work for our research consists of three areas: self-organization, multi-agent systems, and monitoring.

A. Self-Organization

Lots of existing systems from various fields have self-organizing capabilities [3][5]. In biology for example, the neural and hormonal systems in living beings are self-regulated, many animals tend to organize themselves within colonies, the human immune system has the ability to react to new threats, although it has not been specifically planned to handle them [5][18]. Self-organization is achieved by different mechanisms facilitating cooperation, e.g., stigmergy, feedback, or heart-beats.

One of the most famous cooperation mechanisms existing in nature is the *stigmergy* principle, which can be observed in ant colonies [3]. Stigmergy is a form of indirect communication between agents via their environment. Ants cooperate in this way by releasing and sensing pheromones.

Another widely used mechanism for self-organization is the *feedback* concept [3][5]. *Positive feedback* is used as a way to amplify changes in systems. In order to avoid endless and uncontrollable amplifications, *negative feedback* is used in combination and can be seen as a counteracting mechanism aimed at stabilizing systems.

The *heart-beat* mechanism is a form of cooperation which is used for local monitoring [12][13]. In biological organisms, cells monitor each other in a decentralized manner by sending heart-beats periodically to other cells. Heart-beats are typically very simple messages, containing almost no information, since their only goal is to notify neighbours that their sender is alive. Heart-beats with more information are named *pulses*. Pulse monitors are used as reflex signals to give urgent warning about an undesirable situation [18].

B. Multi-Agent Systems

When it comes to cooperation (and more generally to self-organization) mechanisms, agents are the basic entities these mechanisms are based on. *Multi-Agent Systems* (MAS) are sets of agents interacting with each other. As stated by Gabbai et al. [6], in MAS more attention is paid on interactions than on the agents' individual actions. Agents composing an MAS accomplish complex global tasks, while they have limited individual capabilities and operate only at a local level [20]. Since global tasks at the system level are based on interactions between autonomous agents, the latter have to rely on each other to execute their own actions. However, some agents in the system may be malicious or experience problems. The theory of *reputation* in Multi-Agent Systems has been developed in order to support agents in their interactions with their peers. Here, reputation is the perception one agent has of another. It can be modelled centralized or decentralized, i.e., storing reputation information among the agents. Agents evaluate each other and use this perception of the others to decide which actions to perform.

C. Monitoring

In order to detect and then to react to violations of SLAs, the fulfilment of these contracts has to be monitored during runtime. Monitoring approaches can be divided into centralized and decentralized ones.

Many *centralized* approaches mainly deal with service monitoring and not with the reactions to detected problems [4][17][19]. Furthermore, centralization is here often rather limiting and can lead to scalability and performance problems.

For these reasons, we use decentralized monitoring for our concept. Decentralized monitoring approaches do not rely on a central entity, as they make use of distributed data gathering and distributed decision making. Approaches for decentralized monitoring are generally based on agent technology. In the field of service-based workflows, Zeng et al. have proposed a management system based on agents [21] integrating dynamically and effectively cross-enterprise workflows.

Our approach is built on the *AMAS.KOM* architecture (Automated Monitoring and Alignment of Services), which is a decentralized monitoring and deviation handling architecture based on agent technology [16]. This architecture has been designed to monitor Web service-based workflows and to support the handling of SLA violations autonomously. Fig. 2 gives a simplified overview of the AMAS.KOM architecture. In this monitoring infrastructure, Web service calls are redirected by proxies to Monitoring and Alignment Agents (MAAs). Each agent is used as a monitoring unit for a given Web service with a given SLA. MAAs then act as proxies for their assigned Web services, while monitoring them and checking the fulfilment of the associated SLAs.

III. Basic Concepts and Implementation

The details of our agent cooperation concept have been published previously [11]. However, this section is a necessary foundation for our evaluation approach and briefly recapitulates the major aspects.

A. Agent Cooperation Concept

In the AMAS.KOM architecture, one Monitoring Agent is monitoring the fulfilment of one SLA for one Web service. Obviously, a Web service is not dedicated to only one client. To use Web service technology to its full extend, a Web service serves several service consumers concurrently. As a result, a Web service may be subject to several contracted SLAs and, thus, be monitored by several agents simultaneously, each agent monitoring the fulfilment of one SLA in particular.

Using all the agents monitoring the same Web service we create a grouping, called an *Agent Cluster* as the

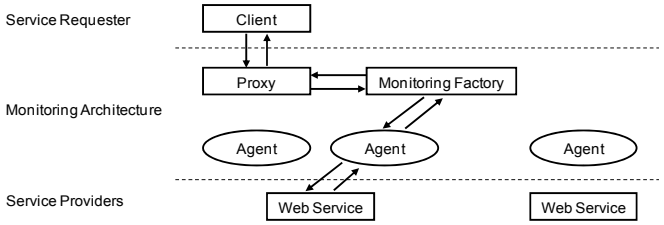

Fig.2 Simplified overview of the AMAS.KOM architecture [11]

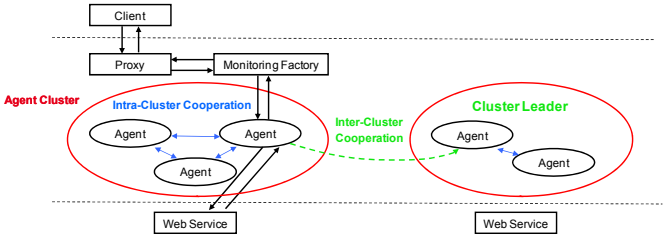

Fig.3 Overview of the Agent Cluster concept [11]

foundation for further cooperation. Fig. 3 gives an overview of this clustering concept. Within this cluster, the agents exchange their monitoring information and, thus, as a whole, the cluster is capable of specialized diagnosis. It can distinguish a Web service crash from the inability to fulfil an SLA, or from problems in the monitoring infrastructure.

With this more accurate perception of the monitored Web service, the Agent Cluster is able to perform suitable reactions to detected violations autonomously. These reactions may include, e.g., the invocation of alternative Web services, the delegation of monitoring to other agents (which may be generated for this purpose), and the renegotiation of SLAs. As a result, while a single agent is responsible for the monitoring of the fulfilment of an SLA contracted by a given Web service, the associated Agent Cluster is, as a whole, in charge of monitoring this Web service and handling the detected deviations. The cluster accomplishes these tasks on its own without external intervention. In addition, the cluster manages itself by treating potential problems occurring in the agent infrastructure inside the Agent Cluster.

For tasks in the Agent Cluster that could better be centralized, one agent is elected among the Monitoring Agents in the cluster: the *Cluster Leader*. This is a concept known, for example, from hierarchical routing protocols in Wireless Sensor Networks [1] or the Cougaar architecture [7]. After its election, the Cluster Leader still acts as a regular Monitoring Agent, but it has additional responsibilities, e.g., receiving requests from other Agent Clusters. It has to decide whether to accept them, according to its perception of the Web service it monitors as well as its perception of its Agent Cluster. In case that it accepts such external requests, the Cluster Leader treats them by delegating them to the Monitoring Agents in its cluster.

As its main task, the Cluster Leader is responsible for the management of its Agent Cluster. To achieve this, the Cluster Leader is the centre of the heart-beat mechanism inside the cluster. Periodically, it sends heart-beats to the other Monitoring Agents in the cluster to notify them of its presence. In their turn, the other agents send heart-beats back (cf. Fig. 4). In this way, the Cluster Leader is aware of the presence of all the Monitoring Agents inside its cluster

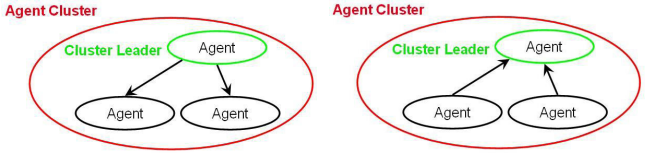

Fig.4 The Cluster Leader sends heart-beats to all the agents in the cluster (left) and receives heart-beats in return (right)

and can detect agent failures. In such a case, the leader tries to delegate the monitoring tasks of the problematic agent to another one in the cluster, or it creates a new agent which will replace the damaged one and take over its tasks.

The Cluster Leader stores only some basic information about the other agents in the cluster, thus, its storage is no point of failure. In case of failure of the Cluster Leader, a dedicated election mechanism ensures the election of a new leader which will retrieve and regenerate this information as soon as it is elected.

In general, choosing one agent per cluster to treat incoming communication from other Agent Clusters is a scalable way to design *inter-cluster cooperation*. Moreover, electing one agent per cluster for extra responsibilities is a robust way of dealing with central non-critical functionality in the clusters (*intra-cluster cooperation*).

B. Implementation Details

Building on the original AMAS.KOM architecture, our prototype with the cooperation enhancements discussed in this paper is implemented using the JADE (Java Agent DEvelopment) framework [2], an Open Source Java framework for agent development compliant with the FIPA specification (Foundation for Intelligent Physical Agents). To ensure extensibility and reusability, special care has been taken of the proper definition of new behaviors in JADE in order to enable future work.

Apache AXIS2 is used for Web service deployment and the following Web service standards are supported by the original as well as by the enhanced architecture: WS-BPEL 2.0 for workflow description, WSDL 1.1 for Web service description, SOAP 1.2 as the protocol for exchanging XML messages, REST as the style for Web service communication, and WS-Policy and WS-SecurityPolicy as policy formats.

IV. Concept Evaluation

This section discusses the evaluation of the previously described agent cooperation concept. It is structured into the description of the used methodology and the presentation of the obtained results.

A. Methodology and Infrastructure

In order to evaluate our agent cooperation concept, two general methods are possible. The evaluation could either be performed by upgrading the implemented architecture to make it support simulations, or by using a special simulator, e.g., OMNeT, running separately from the developed prototype. The second evaluation process has one great drawback compared with the first one: the performed simulations would be totally independent from the implemented architecture. Obtaining good simulation results this way would not guarantee at all that the implemented concept runs effectively. In addition, this method would require re-implementing the concept for the simulator, possibly introducing new sources for errors. Therefore, we decided to evaluate the agent cooperation

Fig.5 Simulation interface for the evaluation of our concept

concept by using the implementation sketched in the previous section. In order to support simulations and to facilitate the handling of the results, a simulation interface has been developed and integrated into our prototype. The basic idea is to have one user interface for each Monitoring Agent created during run-time. A screenshot of such an interface is shown in Fig. 5.

The interface for an agent is used on the one hand to display logging information for this agent and on the other hand to enable the user performing the evaluation to modify certain simulation parameters. These parameters include the *reachability of the Web service* from the agent, the *latency* of the Web service from the agent's point of view, and the *reachability of the agent* itself from the rest of the cluster's point of view. By interacting with the simulation interface, the user is able to simulate the different types of causes for SLA violations, which are described in the next section as our evaluation use cases.

B. Qualitative Evaluation of Our Agent Cooperation

In this section, the different types of SLA violations for which the agent cooperation concept has been designed are presented as use cases:

- *Web service problems* are simulated by decreasing the value of the parameter "WS Reachability" or by increasing the value of the parameter "Latency in WS calls" for each agent monitoring this Web service.
- *Connectivity problems* between an agent and its Web service are simulated by decreasing the value of the parameter "WS Reachability", which causes service failures, or by increasing the value of the parameter "Latency in WS calls", which increases the service's response time.
- *Agent problems* are simulated by increasing the value of the parameter "Agent Reachability", or by even destroying the agent (closing the interface window).
- *Simulating a too optimistic SLA* is realized by setting a very small value for the maximum response time in the policy file corresponding with this SLA (which is passed to the agent at its creation time).

In addition to the use cases above, the "*regular case*" has to be considered where no SLA violation occurs. The goal here is to prove that the basic mechanisms the whole architecture is based on run effectively, i.e., the creation of Monitoring Agents, the "yellow pages" service, the intra-cluster releasing feedback according to the "Publish-Subscribe" pattern, the heart-beat, and the election mechanisms.

The implemented mechanisms were evaluated *qualitatively* for each use case, using the simulation interface presented in the previous section. Using the information gathered from the logs displayed in the user interface, *sequence diagrams* were drawn in order to give a better overview of the interactions occurring between the different entities of the monitoring architecture, i.e., showing the effectiveness of the used mechanisms. Due to the severe space constraints, in this paper only one of the use cases is presented in detail. (Further results are available from the authors upon request.)

The main use case for the presentation of our evaluation is when the monitored *Web service experiences problems*. All agents monitoring the Web service detect SLA violations, e.g., too high response times or failures of the service calls. Both cases are evaluated in the following:

Failure of Service Calls: In case of failures of service requests due to Web service problems, the other agents of the cluster also detect violations. The Monitoring Agent should not contact other agents in its cluster for the results of the requests, but it should contact the Cluster Leader of an alternative Web service. After experiencing several failures, it should delegate its monitoring tasks to an agent of another Agent cluster associated with an alternative Web service.

The following describes what happens in detail, for better differentiation, we use numbers for the participating agents and Web services: after experiencing a failure, Agent 1 sends negative feedback to its cluster. In response, the other agents check the Web service, experience failures in their turn, and release negative feedback into the cluster as well. After receiving these feedback messages, Agent 1 decides to contact Agent 5, Cluster Leader of the alternative Web service Web service 2, for the result of the service request. Afterwards, Agent 1 considers the current situation as satisfying and decides to continue in this way. After experiencing a new failure and receiving negative feedback from the other agents of the cluster, Agent 1 requests once again Agent 5 for the result of its service request. Afterwards, it decides to delegate its monitoring tasks to an agent of an alternative Web service by sending a call for proposals. The Cluster Leader Agent 5 makes a proposal which is accepted by Agent 1. It creates a new agent Agent 1' which takes over the tasks of Agent 1 and monitors Web service 2. Finally, Agent 1 dies. When there is no alternative Web service, inter-cluster requests are not possible and the agent finally has to return a failure message. Without an alternative Web service, it cannot delegate its tasks to another agent and must go on this way. The sequence diagram in Fig. 6 depicts this special case.

Slow Web Service: After experiencing several simple SLA violations because of the high latency of the Web service, the Monitoring Agent should delegate its monitoring tasks to an agent of another Agent Cluster, which is associated with an alternative Web service. This use case is equivalent to the previous one without the inter-

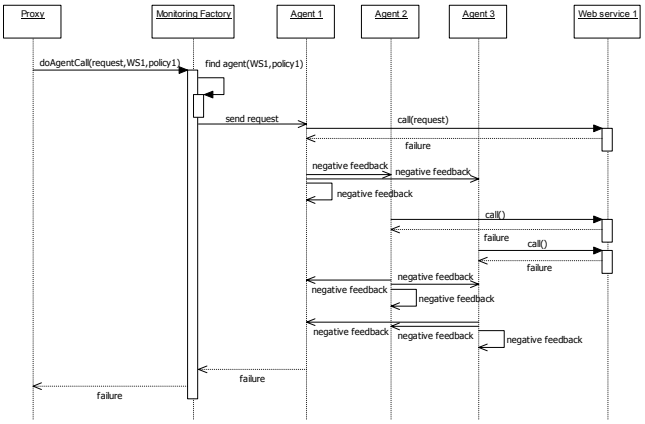

Fig.6 Web service problem: failure of service call (without alternative Web Service)

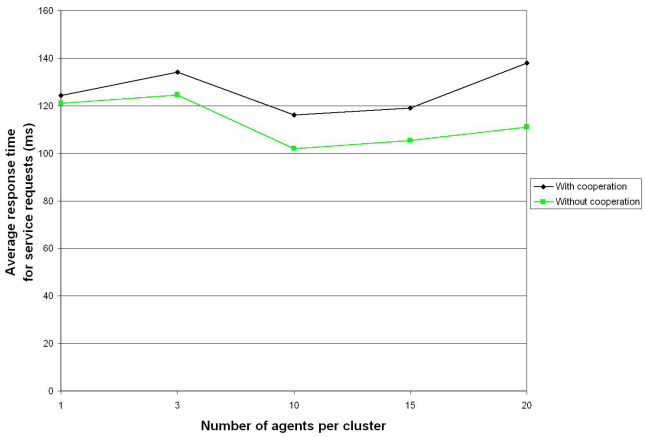

Fig.7 Comparison of response time for service requests with and without cooperation

cluster requests for results.

C. Quantitative Impact of the Cooperation Mechanisms

Our agent cooperation model was designed with the objective of detecting and handling SLA deviations. In the preceding section, we have shown our implementation of the concept to deal with these violations successfully.

However, it seems obvious that the cooperation mechanisms integrated into the AMAS.KOM architecture introduce communication overhead in the regular case, i.e., when no deviation occurs. The exchange of messages between agents enables them to react to violations successfully, but it has no effect otherwise. Therefore, we discuss in this section the overhead measurements of the cooperation mechanisms in the regular case. We evaluate this overhead based on the response time for service requests on the client side, i.e., the time for the client to obtain results for service requests. This metric can be seen as the most relevant one for this case since the agent cooperation concept is aimed at offering advantages to the service customers. The comparison of the response time with and without cooperation is a way to check its impact on the clients, i.e., performance drawbacks.

In addition, this comparison is closely related to the number of messages between the agents for cooperation, since these messages induce some latency into the system. As most of the exchanged messages between agents come from intra-cluster cooperation, the amount of these messages mainly depends on the size of the clusters.

Therefore, we modify the number of agents per cluster in our experiment. For each cluster size, we send 10 service requests to each Monitoring Agent and we average the response time. Fig. 7 shows the results of this experiment. Response times with and without cooperation are compared, for cluster sizes ranging from 1 to 20. Fig. 8 represents the relative difference between the two cases. As expected, the overhead due to the exchange of messages between agents for cooperation increases with the cluster size. This is expected since the number of messages exchanged between the agents in the cluster (feedback, heart-beats) depends on the size of the Agent Cluster. For bigger clusters, e.g., with 25, 30, or 50 agents, the response time increases in a serious way, leading to client timeouts, i.e., 30 seconds with Apache Axis2 while response times are around 100 milliseconds without cooperation. After carefully checking the logs, we noticed that in a bigger cluster, elections are launched all the time. The Cluster Leader has to manage more heart-beat messages, while the defined timeout for election launching (15 seconds) is too small to allow it to cope with all these messages. Therefore, flooded by the high amount of heart-beat messages as well as dealing with the service requests and the feedback messages, the Cluster Leader cannot handle these heart-beats within the election timeout. An election is then launched. As result, a newly elected leader has the same difficulties as the former one, another election is launched and so on.

Consequently, the Agent Cluster experiences *endless election launching*. As election messages contain the agents' perception vectors and are quite big messages, they overflow the whole JADE agent platform. Not only the Cluster Leader but also the other agents cannot treat their service requests in time. In order to prevent this endless election launching process, one possibility is to increase the timeout for election launching. Even for smaller clusters, increasing the different timeouts in the system, e.g., the leader's period of sending heart-beats, would also reduce the overhead (by reducing the number of heart-beats). However, by increasing the timeouts, the system gets less

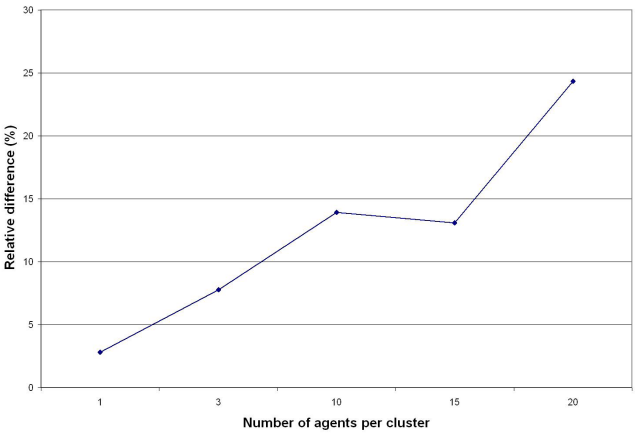

Fig.8 Relative difference of response time for service requests with and without cooperation

reactive. If the current election timeout may be too small (15 seconds) and could be increased, this can be done only up to a certain extent. Anyway, for each timeout value, there will be cluster sizes for which the above problem will occur.

Therefore, we propose to limit the size of Agent Clusters to prevent this problem, e.g., 15 agents per cluster. If too many agents monitor the same Web service, it will be decided to create other Agent Clusters for this Web service. This way, a Web service will not be assigned only one Agent Cluster, but could be associated with several ones. Special inter-cluster cooperation will be required to achieve this, i.e., between the leaders. This opens opportunities for future work, as described in the next section.

V. Conclusions and Future Work

In this paper, we presented the evaluation of our agent cooperation model which focuses on monitoring and handling SLA deviations autonomously and which enhances an existing monitoring architecture for cross-organizational, Web service-based workflows.

As a proof of concept, we implemented our agent cooperation model and integrated it into the original AMAS.KOM architecture.

For evaluation purposes, we developed a simulation interface and integrated it into the enhanced architecture. The different types of SLA violations considered in our scenario were presented as use cases and the concept was qualitatively evaluated for each type of violation. One special use case was discussed in detail. This way, we proved that our implementation of the agent cooperation concept made Monitoring Agents handle SLA violations suitably. While our cooperation model brings benefits to the monitoring architecture by enabling it to react to detected deviations, we also had to investigate its impact on the operation of the architecture in the regular case, i.e., when no violation occurs. For this purpose, we measured the overhead due to the cooperation mechanisms in the regular case. We observed that the overhead increased with the cluster size, which was expected since the number of messages exchanged between the agents depends on the size of the Agent Cluster. Therefore, an important step is to limit the number of agents within a cluster, using even multiple clusters for the monitoring of the same service, if necessary.

Other levels of cooperation could be added and other functionalities could be integrated into the monitoring architecture: in our concept, inter-cluster cooperation remains limited to requests for results and calls for delegation. Inspired by hierarchical routing in Wireless Sensor Networks [1], we could imagine some exchange of information between leaders of different clusters. As we finally decided to limit the cluster size and then to allow the assignment of several Agent Clusters per Web service, special cooperation mechanisms between Cluster Leaders assigned to the same Web service have to be considered.

VI. Acknowledgments

This work is supported in part by E-Finance Lab e.V., Frankfurt am Main, Germany and BearingPoint Management and Technology Consultants.

VII. References

[1] J.N. Al-Karaki and A.E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey," *IEEE Wireless Communications*, vol. 11, no. 6, 12, 2004, pages 6–28.

[2] F.L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, Wiley, 2007.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, Inc., New York: 1999.

[4] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes," in *Proceedings of the International Conference on Service Oriented Computing*, 2005, pages 269–282.

[5] S. Camazine, N.R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, and G. Theraulaz, *Self-Organization in Biological Systems*, Princeton University Press, Princeton: 2001.

[6] J.M.E. Gabbai, H. Yin, W.A. Wright, and N.M. Allinson, "Self-Organization, Emergence and Multi-Agent Systems," in *ICNN&B'05: The 2005 IEEE International Conference on Neural Networks and Brain*, 2005, http://gabbai.com/files/ICNNB.

[7] A. Helsinger and T. Wright, "Cougaar: A Robust Configurable Multi Agent Platform," in *IEEE Aerospace Conference*, 2005.

[8] N. Josuttis, *SOA in Practice: The Art of Distributed System Design*, O'Reilly Media, Inc., 2007.

[9] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices,* Prentice Hall PTR, Upper Saddle River: 2004.

[10] H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck, *Web Service Level Agreement (WSLA) Language Specification*, version 1.0, technical report, IBM Corporation, 2003.

[11] A. Miede, J.-B. Behuet, N. Repp, J. Eckert, and R. Steinmetz, "Cooperation Mechanisms for Monitoring Agents in Service-oriented Architectures," in *Tagungsband der 9. internationalen Tagung Wirtschaftsinformatik*, 2009, volume 1, pages 749–758.

[12] A. Miede, N. Repp, J. Eckert, and R. Steinmetz, "Self-Organization Mechanisms for Information Systems - A Survey," in *Proceedings of the Fourteenth Americas Conference on Information Systems*, 2008.

[13] R. Nagpal, "A Catalog of Biologically-Inspired Primitives for Engineering Self-Organization," in *Engineering Self-Organising Systems, Volume 2977 of Lecture Notes in Computer Science*, G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli (editors), Springer, 2003, pages 53–62.

[14] E. Newcomer and G. Lomow, *Understanding SOA with Web Services*, Addison-Wesley Professional, 2004.

[15] M.P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," in *Proceedings of the 4th International Conference on Web Information Systems Engineering*, 2003, pages 3–12.

[16] N. Repp, J. Eckert, S. Schulte, M. Niemann, R. Berbner, and R. Steinmetz, "Towards Automated Monitoring and Alignment of Service-based Workflows," in *IEEE International Conference on Digital Ecosystems and Technologies,* 2008.

[17] A. Schmietendorf, R.R. Dumke, and S. Stojanov, "Performance Aspects in Web Service-based Integration Solutions," in *21st UK Performance Engineering Workshop*, 2005, pages 137–151.

[18] R. Sterritt and M.G. Hinchey, "Biologically-Inspired Concepts for Self-Management of Complexity," in *ICECCS '06: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems*, 2006, pages 163–168.

[19] G. Spanoudakis, C. Kloukinas, and K. Androutsopoulos, "Towards Security Monitoring Patterns," in *SAC '07: Proceedings of the 2007 ACM Symposium on Applied Computing*, 2007, pages 1518–1525.

[20] J.M. Vidal, *Fundamentals of Multiagent Systems: Using NetLogo Models*, 2006, http://www.multiagent.com/fmas/.

[21] L. Zeng, A. Ngu, B. Benatallah, and M. O'Dell, "An Agent-based Approach for Supporting Cross-enterprise Workflows," in *ADC '01: Proceedings of the 12th Australasian database conference*, 2001, pages 123–130.