MBR+09

[MBR+09] *André Miede, Jean-Baptiste Behuet, Nicolas Repp, Julian Eckert, Ralf Steinmetz;* **Cooperation Mechanisms for Monitoring Agents in Service-oriented Architectures.** In: Hans Robert Hansen, Dimitris Karagiannis, Hans-Georg Fill: Tagungsband der 9. internationalen Tagung Wirtschaftsinformatik 2009, vol. 1,Österreichische Computer Gesellschaft, February 2009.Seite 749–758.

COOPERATION MECHANISMS FOR MONITORING AGENTS IN SERVICE-ORIENTED ARCHITECTURES

André Miede, Jean-Baptiste Behuet,
Nicolas Repp, Julian Eckert, Ralf Steinmetz[1]

Abstract
The Service-Oriented Architecture paradigm (SOA), e.g., realized with Web Services technology, enables enterprises to establish cross-organizational, service-based workflows. An important issue is the monitoring of the fulfillment of Service Level Agreements (SLAs) which define the responsibilities between the participants. Recent research has shown that agent technology is a useful approach in this context. Thus, we present ways for agent cooperation on different levels of abstraction. This cooperation aims at monitoring workflows and especially to react to deviations in different scenarios of SLA violations.

1. Introduction

In an international and highly competitive economy, modern enterprises face many challenging requirements. To address these challenges, both the technology side and the business side have to cooperate seamlessly, while maintaining a mutual understanding of the challenges and their possible solutions on both sides. Among the different requirements which affect existing and future enterprise *Information Technology* (IT) architectures, the two following have a strong impact on both applications and research [8, 9, 11]:

- Achieving a high flexibility of business processes and their underlying IT.
- The integration of heterogeneous systems.

The *Service-Oriented Architecture* (SOA) paradigm is an option to support an enterprise infrastructure which facilitates the addressed requirements. At the core of SOA is the "service" concept, which has to be understood as the technological representation of business functionality [8]. By using services as flexible components, business processes can be composed from them, abstracting processes from the underlying, mostly monolithic applications and allowing for compositions even across organizational boundaries. Such common and relevant scenarios remain an important focus of application scenarios and research in this field: cross-organizational, service-based workflows.

Establishing cross-organizational workflows is based on a trustworthy and dependable service exchange between the participating parties. *Service Level Agreements* (SLAs) are used to define

[1] Multimedia Kommunikation (KOM), TU Darmstadt

both the responsibilities and requirements of the participants. However, the actually fulfillment or non-fulfillment of the SLAs has to be monitored. Ideally, this is done live at runtime to allow for reactions to detected deviations from the negotiated quality. In order to reduce further complexity and new, costly layers of manual administration, concepts from the field of self-organization (sometimes also known under the name of "Autonomic Computing") can be utilized [10].

This paper presents cooperation mechanisms for agents to monitor service-based workflows in SOAs. These mechanisms are inspired by natural concepts from the field of self-organizing systems and extend an existing decentralized monitoring architecture. The rest of the paper is structured as follows: Section 2 gives an overview of a common application scenario for monitoring workflows, briefly discusses related work, and introduces the monitoring architecture the concept will be built on. Section 3 lays the foundation for the presented concept by discussing the key elements of the cooperation mechanism. Sections 4 and 5 show the core of the cooperation concept, namely intra- and inter-cluster cooperation, which is used to react to deviations from SLAs in different scenarios. Section 6 presents important details about the implementation of the cooperation concept. Section 7 closes the paper with a summary of the key points and an outlook on future work.

2. Scenario and Related Work

2.1. Monitoring and Deviation Handling Scenario

Cross-organizational service-based workflows are commonly described with the following scenario: to orchestrate higher level services and business processes, a service requester – i.e., an enterprise – integrates third-party services offered by various service providers, e.g., business partners. Service Level Agreements are contracted between the two parties, specifying in particular the service availability and the service performance by metrics like response time and throughput [6]. *Figure 1* gives a schematic impression of this setup.

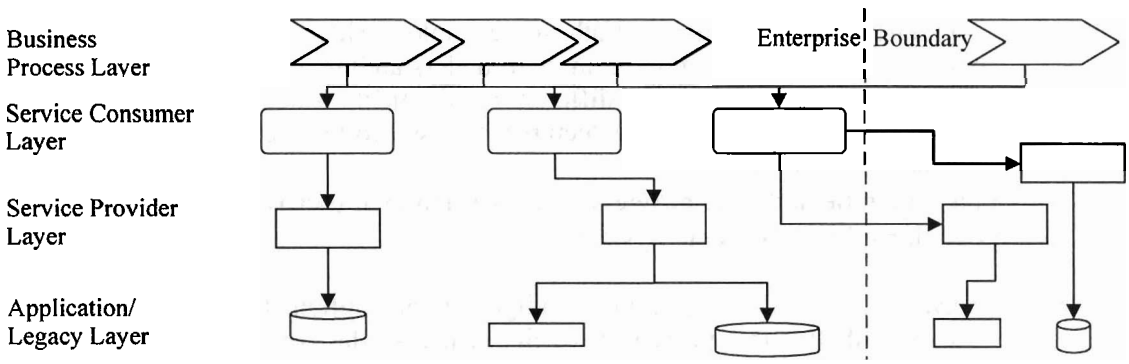

Figure 1: Cross-Organizational, Service-Based Workflows

Just as for any contract between clients and providers, the specification of the SLAs alone is no guarantee by itself. Thus, the monitoring of their fulfillment during runtime is needed as well and the actual service performance and availability have to be compared with the contracted ones, because services may, temporarily or permanently, not be able to fulfill the contracted requirements. Reasons for these failures are diverse as they can be caused by changes in the implementation of the services, overloads due to poor resource planning, crashes and outages of the service providers, or improperly negotiated SLAs that cannot be fulfilled by the services [3].

On the other hand, the monitoring infrastructure itself may also experience problems. In this case, what at first sight was detected as an SLA violation, is none and thus should not be treated as one. More detailed information has to be gathered to support decisions for differentiated reactions. Based on the gathered monitoring information, deviations from the SLAs or infrastructure problems have to be handled accordingly, so that the workflows meet the requirements again. Reactions from such violations may include the renegotiation of the SLAs, the invocation of alternative services, the creation of new monitoring units, etc. However, in this scenario it is important to identify the cause of the problem as specific as possible to react the right way and closest to the failure.

2.2. Related Work

Approaches for monitoring and alignment of service-based workflows can be classified into centralized and decentralized ones, and into functional and non-functional monitoring.

Centralized monitoring and alignment means centralized gathering of monitoring data and centralized decision making, while the collection of monitoring data itself may be realized in a distributed manner by monitoring probes. Baresi and Guinea proposed such an approach, based on a central Monitoring Manager [1]. Although the possibility of recovery actions is mentioned, the authors did not provide any solution for deviation handling.

To cope with scalability and performance problems in large-scale SOAs, centralized approaches are limited. Solutions have been proposed for distributed monitoring based on mobile agent technology. Most of the proposals for distributed monitoring are intended for network management, like [4]. But the agent technology has also been proved to be suitable for automated and decentralized monitoring of services. The *AMAS.KOM* architecture has been developed for this purpose [12, 13]. In the AMAS.KOM architecture (cf. *Figure 2*), service calls are redirected by proxies to agents which act as monitoring units. Agents are then used as proxies for services, monitoring them, checking the fulfillment of SLAs, and being able to take countermeasures in an autonomic fashion.

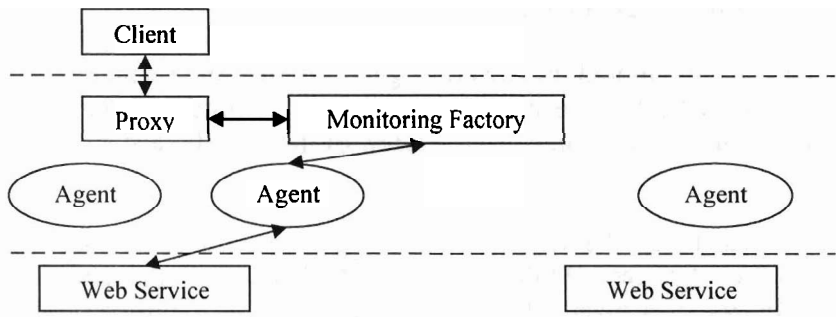

Figure 2: Simplified Overview of the AMAS.KOM Architecture

The generation of *Monitoring and Alignment Agents* (MAA) is automatically done based on the workflow description and the corresponding business requirements. This generation is realized through a transformation of the workflow description and the business requirements into monitored instances of the workflow. The transformation process contains four steps:

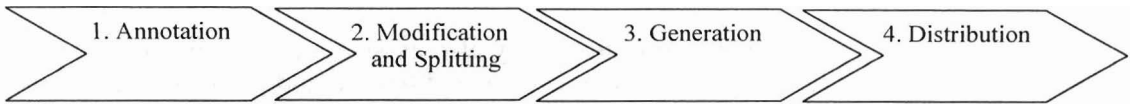

Figure 3: Workflow Transformation Steps

In the *Annotation* step, the business requirements are described for the complete workflow in a single policy document in machine-readable format. In the *Modification and Splitting* step, requirements for each service of the workflow are derived from the global policy document. The MAAs are created during the *Generation* step based on these derived requirements. As this step is especially relevant for the understanding of the architecture and the presented concept, more details are necessary: An MAA associated with an SLA for a given Web Service is generated and started by the central *Monitoring Factory* for the first call to this Web Service with the given SLA. The required configuration parameters are passed during the agent construction, i.e., which Web Service to monitor, the used SLA, specifications for handling deviations, etc. The content of the request to the Web Service itself is then sent to the Monitoring Agent subsequently. For further calls to the same Web Service with the same SLA, the reference to the responsible agent will be retrieved and then the request will be sent to the agent. Finally, the agents are distributed in the infrastructure during the *Distribution* step.

The AMAS.KOM architecture has been designed to support automated monitoring and alignment of service-based workflows. Currently, monitoring capabilities have been integrated into the agents, which now work individually, forwarding service calls, and checking the fulfillment of the contracted SLAs. In order to increase the architecture's abilities to react autonomously to different types of deviations, cooperation mechanisms among the agents can be considered the next necessary step. The current AMAS.KOM architecture is enhanced by integrating these cooperation mechanisms into the monitoring agents. Details of this concept are described in the following.

3. Cooperation Concept Overview

3.1. Hybrid Architecture – Agent Clusters

A Web Service may be monitored by several agents at the same time, each monitoring the fulfillment of a contracted SLA. It seems therefore natural to group all the agents monitoring a given Web Service into an *Agent Cluster*. An overview of the overall cluster concept is provided in *Figure 4*.

An Agent Cluster consists of the agents monitoring a specific Web Service and therefore corresponds to this Web Service. While each individual agent of the cluster is in charge of monitoring the fulfillment of a given SLA by the Web Service, a cluster of agents cooperating together by exchanging monitoring information has a better and more detailed view of the monitored Web Service. Therefore, such a cluster of cooperating agents has special diagnosis capabilities and can distinguish a Web Service crash from a simple inability to fulfill the contracted SLA, or from problems actually occurring inside the agent infrastructure. Consequently, it can determine suitable reactions to overcome the current problems and execute them in an autonomic way, e.g. invocation of alternative Web Services, SLA renegotiation, delegation of the monitoring to other agents, or other reactions.

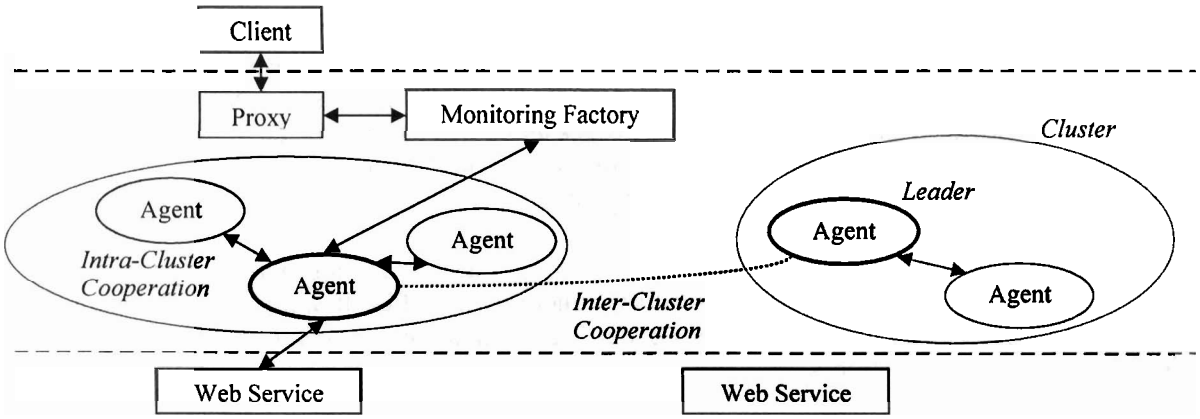

Figure 4: Overview of the Cooperation Concept

As a result, the Agent Cluster associated with a given Web Service is responsible for the monitoring of this Web Service and for the handling of many different types of deviations.

3.2. Distributed Approach Inside the Agent Cluster Based on Monitoring Agents

The agents forming an Agent Cluster are basically all *Monitoring Agents*. A Monitoring Agent is in charge of monitoring the fulfillment of a given SLA by the Web Service. Web Service calls associated with a given SLA are redirected by proxies to the corresponding agent. This Monitoring Agent acts as a proxy for the Web Service, forwarding the requests to the Web Service, while monitoring it and checking the fulfillment of the associated SLA. Monitoring Agents are also capable of cooperation (cf. Section 4) and of taking countermeasures in an autonomic fashion.

For tasks other than monitoring, such as the diagnosis of SLA violations based on the monitoring information from the different agents, the execution of countermeasures, or cluster management, it was first considered to add task-specialized agents into the Agent Clusters. For example, adding one "Diagnosis Agent" aggregating the monitoring information from the monitoring agents of the cluster and which is therefore capable of diagnosis. However, such an approach would require the presence of a multitude of task-specialized agents per cluster, and thus per Web Service, imposing a lot of overhead on the architecture. Moreover, such specialized agents with central functions would constitute critical points-of-failure inside the cluster.

Since the number of Monitoring Agents per cluster (equal to the number of agreements contracted by the associated Web Service) is limited, there is no need for specialized agents, e.g., aggregating the monitoring information. An "*n* to *n*" communication pattern inside the Agent Cluster is fully scalable, and much more robust. Our proposal is therefore a distributed approach inside the Agent Cluster, based solely on Monitoring Agents and on their cooperation, as described in Section 4.

3.3. Elected Cluster Leader

For some tasks in the cluster, which cannot be decentralized, an agent is elected among the Monitoring Agents: the *Cluster Leader*. Once elected, the Cluster Leader still performs its regular duties just as any Monitoring Agent, but it also takes extra responsibilities in addition.

The Cluster Leader is especially the interface of the Agent Cluster for the agents of other clusters. It is in charge of the incoming communication from other Agent Clusters. If another Web Service

crashes, an agent monitoring this service will try to delegate its requests to an alternative Web Service. For this purpose, the latter agent may contact the corresponding Cluster Leader and ask for treating its requests. The Cluster Leader is responsible for receiving these requests and to forward them. It has to decide whether to accept such requests or not, according to the performance and other monitoring criteria of the Web Service it monitors. The Cluster Leader treats those external requests by delegating them to the Monitoring Agents of its cluster.

Electing an agent among the Monitoring Agents of the cluster for taking extra responsibilities is a robust way of dealing with central non-critical functionalities like treating requests from other Agent Clusters. Details on how the Cluster Leader is elected and how it can be replaced in case of a failure are described in Section 4.

4. Intra-Cluster Cooperation

4.1. Exchange of Monitoring Information

The communication pattern for the exchange of monitoring information between the agents inside a cluster is inspired by the self-organization mechanism *Stigmergy*, a communication style observed in nature, especially in ant colonies [2, 10]. Stigmergy is a form of indirect communication between agents by modifying their environment. It is a self-organization mechanism enabling very simple agents to create complex structures, like ants creating foraging networks or building nests only by releasing pheromones. Likewise, Monitoring Agents exchange their information in a distributed way by releasing positive and negative feedback in the Agent Cluster. After a Web Service call, a Monitoring Agent releases negative feedback if it detects an SLA violation or experiences a failure, and releases a positive feedback otherwise. Feedback consists typically of simple messages and may contain some information like the response time of the last service call, possibly the content of the service request, and in the case of a negative feedback whether it refers to a SLA violation or a failure.

The other agents of the cluster modify their behaviors according to received negative feedback. In turn they check the Web Service if they have not done so lately and release feedback. While sending and receiving feedback, agents update their perception of the monitored Web Service (the Web Service's "*reputation*"). In order to avoid a chain reaction of feedback releases, agents do not react to further negative feedback after a certain time. A Monitoring Agent reacts to negative feedback from other agents in the cluster with top priority since such feedback may suggest forthcoming problems for the agent itself.

4.2. Handling of SLA Violations

After Monitoring Agent detects a deviation from the requirements, it releases negative feedback in its cluster. After that, it correlates the monitoring results from the other agents that it has received as feedback, updates its perceived reputation of the service, performs a simple diagnosis and determines whether the detected SLA violation actually comes from an inability for the Web Service to fulfill these requirements, a service crash, a problem in the agent infrastructure, or an isolated accident. Based on this simplistic diagnosis, the agent decides to delegate the monitoring of its SLA to another agent in the cluster, or to an alternative Web Service. Delegation to an alternative Web Service is performed when the agent perceives that the original service is unable to fulfill the contracted SLA, the mechanism is explained in Section 5.

In the case when its latest service call has failed, the Monitoring Agent first needs to get the result of that request as soon as possible. It requests this result from another agent in the cluster or an alternative Web Service. The choice of the agent in the cluster or the alternative Web Service to delegate to is based on the current perception of these (see agent reputation in Section 4.3 and service reputation in Section 5). In the cases of both a simple SLA violation and failure, Monitoring Agents may need to delegate to other agents or alternative Web Services for treating their further requests and fulfilling the SLAs. Such delegations must not be requested as in the case of a failure. Instead, the Monitoring Agents send calls for proposals to other agents in the cluster or to Cluster Leaders of alternative Web Services, which in turn reply with refusals or proposals, specifying their ability to fulfill the given SLA. Based on the received proposals and the current perception of the other agents or Web Services, the requesting agents decide on which services they delegate to. If no proposal is satisfying, i.e. better than the current Web Services, they decide to go on with the current configurations, and possibly ask for the renegotiation of the SLAs.

4.3. Management of the Agent Cluster

The delegation model inside the Agent Cluster, in particular the choice of the Monitoring Agent to delegate to (see Section 4.2), is based on the "reputation" of the agents. The reputation of an agent describes its reliability and its eagerness to cooperate, for example by accepting requests or calls for proposals from other agents and by always releasing feedback. The reputation model we propose for agent reputation is the *Certified Reputation model*, developed by Huynh et al. [5]. This is a promising distributed model which therefore does not need any central agent for managing the reputations.

Besides the choices of the delegating agents, agent reputation is used for the election of the Cluster Leader: the agent in the cluster with the best reputation is elected – if several agents have the same reputation, it is chosen randomly among the best, e.g., taking the one with the smallest id number. Since agents are fundamentally selfish, taking care of their own interests first, their wish for becoming the Cluster Leader and their fear of being excluded from the cluster are good incentives to make them cooperate and to build a good reputation.

The self-management of the Agent Cluster is realized by a *heart-beat mechanism* centralized at the Cluster Leader. The latter sends heart-beats periodically to the agents in the cluster, in order to notify them of its presence. After receiving heart-beats from the leader, the other agents reply by sending heart-beats back. If an agent fails to receive a heart-beat from the leader, it launches an election. In this way, although there is initially no foreseen leader in the cluster, an election is immediately launched. By receiving heart-beats from the other agents, the Cluster Leader is aware of the presence of all the other agents in the cluster, and is able to detect if an agent does not respond anymore. In this case, the leader tries to delegate the tasks of the faulty agent to another agent in the cluster, or in case of necessity asks for the generation of a new agent in the cluster which will overtake the tasks of the faulty one.

5. Inter-Cluster Cooperation

When a Monitoring Agent diagnoses that the Web Service it monitors has crashed or is unable to fulfill the contracted SLA, it may decide to delegate its service requests to an alternative Web Service. An alternative Web Service is a service that implements equivalent functionalities to the original one, but with better Quality of Service characteristics. A list of these alternatives has to be previously defined manually by the client during the SLA negotiation so that it is then given to the

agent in its policy file. In the future, these equivalent Web Services could be discovered and retrieved automatically using semantics.

Monitoring Agents delegate requests to alternative Web Services by contacting their associated Cluster Leaders. In the case when service requests have failed in their own clusters, the Monitoring Agents still need to obtain the results for the current requests as soon as possible. For this purpose, they request the Cluster Leaders of the alternative Web Services for the results to these current requests. The Cluster Leaders treat these requests by delegating them inside their own clusters. Afterwards, as well as in the case of simple SLA violations without failure, Monitoring Agents may need to delegate to alternative Web Services for treating their further requests while respecting the SLAs. Such delegations must not be requested as in the previous case. Instead, the Monitoring Agents send calls for proposals to the Cluster Leaders of the alternative Web Services, which in turn reply with refusals or proposals, specifying the overall "reputations" (see 4.1) of their Web Services. Based on the received proposals, the requesting agents decide on which services they delegate to. If no proposal is satisfying, i.e. better than the current Web Services, they decide to go on with the current configurations, and possibly ask for the renegotiation of the SLAs.

The inter-cluster cooperation is now limited to sending requests or calls for proposals to Cluster Leaders by Monitoring Agents of other clusters. Other cooperation mechanisms, such as communication between the leaders of the different clusters, are currently under investigation.

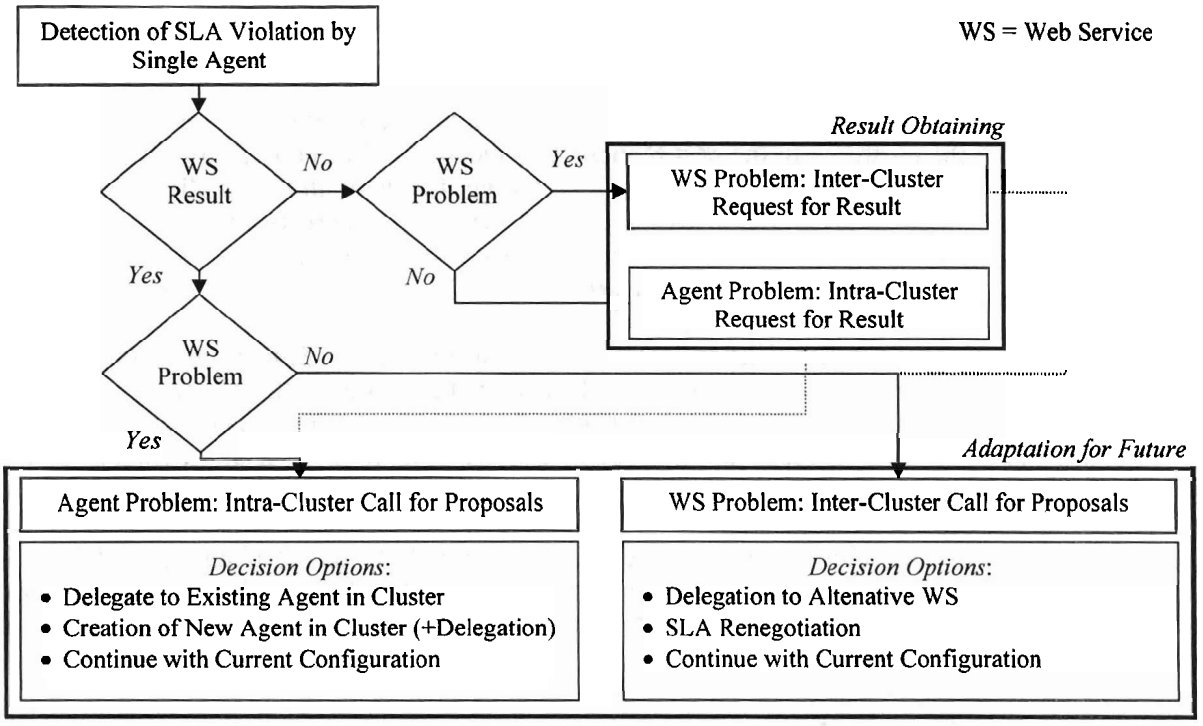

Figure 5: Schematic Overview of the Different Deviation Handling Scenarios and their Interrelations

To sum up the essence of Section 4 and 5, *Figure 5* gives a schematic overview of what deviation handling scenarios can be handled by the proposed collaboration mechanisms.

6. Implementation

The AMAS.KOM architecture and the extensions presented in this paper are implemented using the JADE framework, an Open Source Java framework for agent development, which is compliant with the *FIPA specification* (Foundation for Intelligent Physical Agents) [7]. For Web Service deployment Apache AXIS is used, thus the architecture supports the following common Web Service standards: WS-BPEL 2.0 for workflow description, WSDL 1.1 (Web Service Description Language) for the description of Web Service interfaces, SOAP 1.2 as protocol for the exchange of XML messages, REST as architectural style for service communication, and WS-Policy and WS-SecurityPolicy as policy formats.

The discussed enhancements of the AMAS.KOM architecture consist of adding cooperation capabilities to the monitoring agents. The implementation of these mechanisms is mainly realized by adding new behaviors to the existing agents, using the class *Behaviour* of the JADE framework.

The JADE framework provides the necessary tools for agent development in Java. However, it has to be emphasized that agent development is not identical to object development. One special feature of agent computing is the autonomy of the agents. Autonomic agents cannot be accessed like objects, but they can only be accessed by sending messages to them in the *FIPA ACL format* (Agent Communication Language). For example, the Monitoring Factory consequently has to forward the content of the Web Service requests to the agents as ACL messages, and receives the responses in return as ACL messages as well. Such a communication between objects and agents is provided by the JADE framework, using a *GatewayAgent*.

During the implementation, special attention is paid to the proper definition of new behaviors in JADE in order to allow their future enhancements.

7. Conclusions and Future Work

In this paper, cooperation mechanisms for monitoring and deviation handling agents were presented. Cooperation is built around the idea of special domains, called Agent Clusters, where a set of autonomous agents is monitoring a service for different aspects and reacts to detected problems. Coordination is achieved by the election of a Cluster Leader, which has responsibilities both within the cluster (intra-cluster cooperation) and between clusters (inter-cluster cooperation). This setup makes it possible for the agents to detect deviations and to react to them in a timely manner and according to pre-defined specifications. The concept is an extension of the existing AMAS.KOM architecture, building on its code base using agent technology from the JADE framework.

Future work aims at refining the concept further, especially detailing inter-cluster cooperation to allow for more functionality here and to approach a wider range of deviation scenarios.

Furthermore, the WS-Re2Policy language has to be integrated to have a more flexible and dynamic means to specify what is to be monitored and how should be reacted to deviations [14]. Concerning the implementation, special care has be taken to ensure the maintainability and extensibility of the architecture, e.g., to extend the criteria to be monitored or to extend the available reaction possibilities.

Another very important aspect is the evaluation of the concept and its implementation. Here, issues such as the imposed communication overhead have to be analyzed and weighed against the benefits

of automated monitoring and deviation handling. Due to page restrictions, results for these issues will be presented and discussed in our next publications.

8. Acknowledgments

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

9. References

[1] BARESI, L.; GUINEA, S., Towards Dynamic Monitoring of WS-BPEL Processes, in: Proceedings of the 3rd International Conference on Service oriented computing, 2005, pp. 269-282

[2] CAMAZINE, S.; DENEUBOURG, J.; FRANKS, N. R.; SNEYD, J.; THERAULAZ, G.; BONABEAU, E., Self-Organization in Biological Systems, Princeton Studies in Complexity, Princeton University Press, 2003.

[3] ECKERT, J.; SCHULTE, S.; NIEMANN, M.; REPP, N.; STEINMETZ, R., Worst-Case Workflow Performance Optimization, in: 3rd International Conference on Internet and Web Applications and Services (ICIW'08), Athens, Greece, June 2008.

[4] GAVALAS, D.; GREENWOOD, D.; GHANBARI, M.; O'MAHONY, M., Using Mobile Agents for Distributed Network Performance Management, in: Proceedings of the Third International Workshop on Intelligent Agents for Telecommunication (IATA99), pp. 96-112, 1999.

[5] HUYNH, T.D.; JENNINGS, N.R.; SHADBOLT, N.R., Certified Reputation - How an Agent Can Trust a Stranger, in: The Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, 2006, May 8-12, Hakodate, Japan, 2006.

[6] IBM CORPORATION, Web Service Level Agreement (WSLA) Language Specification, Version 1.0, January 2003.

[7] JADE, Java Agent DEvelopment Framework, http.//jade.tilab.com/, 2008.

[8] JOSUTTIS, N. M., SOA in Practice: The Art of Distributed System Design (Theory in Practice), O'Reilly Media, Inc., 2007.

[9] KRAFZIG, D.; BANKE, K.; SLAMA, D., Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series), Prentice Hall PTR, 2004.

[10] MIEDE, A.; REPP, N.; ECKERT, J.; STEINMETZ, R., Self-Organization Mechanisms for Information Systems -- A Survey, Proceedings of the Fourteenth Americas Conference on Information Systems (AMCIS 2008), 2008.

[11] Newcomer, E.; Lomow, G., Understanding SOA with Web Services (Independent Technology Guides), Addison-Wesley Professional, 2004.

[12] REPP, N., Monitoring of Services in Distributed Workflows, in: 3rd International Conference on Software and Data Technologies (ICSOFT 2008), July 2008.

[13] REPP, N.; ECKERT, J.; SCHULTE, S.; NIEMANN, M.; BERBNER, R.; STEINMETZ, R., Towards Automated Monitoring and Alignment of Service-based Workflows, in: IEEE International Conference on Digital Ecosystems and Technologies 2008 (IEEE DEST 2008), February 2008.

[14] REPP, N.; MIEDE, A.; NIEMANN, M.; STEINMETZ, R., WS-Re2Policy: A Policy Language for Distributed SLA Monitoring and Enforcement, in: Proceedings of the Third International Conference on Systems and Networks Communications (ICSNC), 2008.