Structural Adaptations for Self-Organizing Multi-Agent Systems

Thomas Preisler and Wolfgang Renz

Multimedia Systems Laboratory
Faculty of Engineering and Computer Science
Hamburg University of Applied Sciences
Email: {thomas.preisler,wolfgang.renz}@haw-hamburg.de

***Abstract*—Over one decade of research in engineering of self-organization (SO) has established SO as the decentralized way to build self-adaptive systems. However, such SO systems even when well engineered may, under certain conditions, exhibit unwanted dynamical behavior, e.g. performance may decrease and/or starvation may occur. A promising concept to overcome such dynamical in-efficiencies in SO systems is to realize the dynamic exchange or reconfiguration of the coordination processes responsible for the self-organizing behavior in terms of a *structural adaptation*. In this paper, we propose an architecture and engineering approach to support the self-adaptive, structural exchange (or reconfiguration) of self-organizing coordination processes based on distributed Multi-Agent technology. Here, a sensor in each agent detects any decrease of specified SO performance indicators which initiates a distributed consensus process that allows for the exchange (or reconfiguration) of the self-organizing coordination processes, enabling the system to adapt to changing conditions automatically.**

Keywords–Self-Organizing Systems; Multi-Agent Systems; Structural Adaptation; Decentralized Coordination

I. INTRODUCTION

For self-organizing Multi-Agent Systems (MAS), the capability to adapt to a variety of (mostly external) influences, i.e., their *adaptivity*, is a key feature. In this context, adaptivity describes the ability of a system to change its structure respective behavior in response to external influences or altering demands. In addition, adaptive, self-organizing systems still strive towards reaching (initially defined) global goals. Looking in more detail into such adaptive systems, they reveal many different facets. According to [1] it can be distinguished between *design-time* and *run-time* adaptivity, where the latter one is far more challenging. Figure 1 depicts even more dimensions of adaptive systems. It differentiates between approaches that only change system parameters in order to exhibit adaptive behavior and concepts that can even alter the whole structure respective replace certain components of the system. Furthermore, it discerns between approaches based on centralized or decentralized architectures and on how the adaptivity is managed. Thereby, it is differentiated between solutions where the adaptivity is managed manually or automatically by the system itself. The red dot shown in the Figure ranks the proposed solution according to the different dimensions of adaptive systems. The approach is based on a decentralized architecture and emphasizes structure-based changes (while also supporting parameter-based changes). Possible adaptations are modeled manually at *design-time* and executes automatically at *run-time*. Therefore, the solution is ranked between manual and automatic adaptations.

Developing and operating systems that belong to this class of adaptive systems, i.e., self-organizing systems, is a challenging task. Firstly, it requires a systematic development approach that copes at all stages with three inherent characteristics of these systems: non-linear dynamics, stochastic behavior and emergent phenomena. Secondly, it requires a modular system architecture which enables the adaptation of the structure at runtime using highly customizable and reusable coordination processes. These coordination processes can be understood as standalone design elements that equip a self-organizing system with a specific dynamic behavior. By exchanging these coordination processes at runtime, a distributed application can adapt not only its behavior but also its inherent structure. Thus, enabling the system to overcome problems like performance decrease or starvation, by adjusting the structure of its self-organizing processes automatically. The approach presented in this paper extends already established approaches for engineering self-organizing systems by introducing a system architecture that systematically enables structural adaptations for distributed systems with decentralized control. It is comparable to the reactive planning approach (local) from the BDI (belief, desire, intention) agent architecture [2]. The system as a whole strives towards a distributed consensus (global) to execute predefined structural adaptations plans.

One of the many example of coordination processes in SO systems are distributed consensus algorithms. These algorithms establish self-organization and adaptivity, e.g., in multi-vehicle routing and also exhibit areas in parameter space where they get ineffective and need to get exchanged [3]. In contrast, we will use consensus methods on a meta-control level in this paper in order to detect thresholds for exchanging SO mechanisms (see Figure 2 in Section III).

Accordingly, the remainder of this paper is structured as follows: Section 2 describes related work, followed by Section 3 where the properties and engineering challenges of self-organizing MAS are described and a test scenario is presented. Section 4 introduces the core concept of structural adaptations before Section 5 concludes the paper.

II. RELATED WORK

Current trends in computer science like mobile and ubiquitous computing in combination with an increasing diversification of hard- and software platforms challenge traditional approaches of engineering and operating distributed systems substantially. Years ago, distributed systems were mainly closed systems with a-priori known tasks, challenges, and users. Nowadays, with an increasing pervasiveness of distributed systems, they have turned into an integral part of

 ISBN: 978-1-61208-391-9

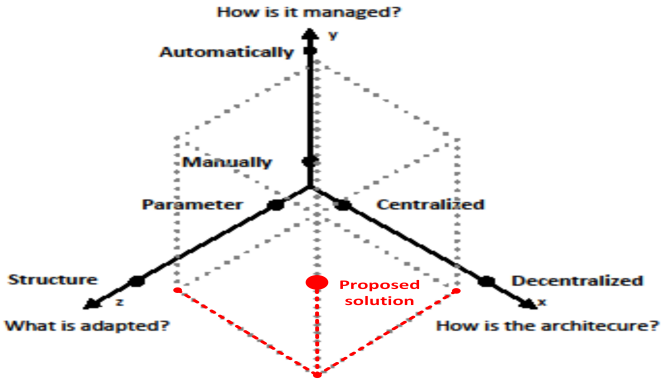

Figure 1. Dimensions of adaptive systems according to [4].

the business world as well as the private life of many people. This evolution implicates new challenges for distributed systems. For instance, they are forced to deal with high and unpredictable dynamics, an increasing complexity, and the satisfaction of non-functional requirements like, e.g., robustness, availability, and scalability. Altogether, this requires a new generation of distributed systems that is capable of *adapting* its behavior *autonomously*. This challenge is addressed by research areas like Autonomic [5] or Organic Computing [6] that aim at providing new approaches to solve it in a systematic fashion. They achieve it with different types of feedback loops, relying on (usually) centralized control elements. The authors of [7] identify feedback loops as the key design element within a distributed system in order to be able to exhibit adaptivity. Here, feedback loops consist of three main components: *sensor*, *actuators* and a *computing entity*. Sensors are in charge of observing the behavior and the (current) status of the system respective the environment it is situated in. Actuators can change the configuration of the system, which can either lead to *parametric* or *structural* changes. The computing entity which serves as a connector between the system input (sensor) and the output (actuator) can be very different w.r.t its internal architecture and abilities [8]. For instance, the widespread autonomic control loop [5], which is based on a monitoring, analyzing, planning, executing and knowledge loop (MAPE-K), contains a centralized computing entity, which can be associated with an autonomic manager to software and hardware components in order to equip them with adaptive behavior. Contrasting to feedback loops, [9] introduces a policy-based approach where the non-functional concerns of an application are described as policies and the application adapts itself to changing conditions controlled by a centralized policy engine. Like the approach presented in this paper [9] also focuses on a clean separation between the business-logic and the self-adapting fulfillment of non-functional requirements.

According to [10], this class of approaches, that introduce centralized control concepts, can be called *self-adaptive systems*. In contrast to this, there are approaches that aim at providing *automatically adaptive systems* which rely on decentralized architectures and utilize distributed feedback loops and coordination mechanisms. They are called *self-organizing systems* [10] and seem, due to their decentralized system architecture, to be better suited to deal with the aforementioned non-functional requirements. Also the concept of self-organization has been observed in many other domains like, e.g., biology, physics, or sociology and has, furthermore, proven its applicability for distributed systems already before (as, e.g., mentioned in [11] [12]).

In addition to the difference between centralized and decentralized feedback loops, another criterion targets the general applicability of existing approaches that aim at providing methods for structural adaptivity for distributed systems. According to [10], adaptivity defines a general system view that can be further decomposed into so called self-* properties of distributed systems. Therefore, there are many approaches which target the provision of a subset of these properties [13] [14] [15]. Whereas these approaches reach good results with respect to specific aspects of adaptive behavior, they lack general methods and concepts that provide structural adaptivity in general. This, however, limits the general applicability of these approaches and forces system developers to deal with (completely) different approaches if there is the requirement for more than just one type of adaptivity. Consequently, if a system requires different self-* properties it has to incorporate different concepts and techniques which increases the complexity of related implementations considerably.

This could be improved by using approaches which are based on structural adaptivity. However, applying them to the concept of self-organization in decentralized systems is an ambitious task. Especially the purposeful engineering of self-organizing systems is challenged by their inherent non-linear dynamics and the bottom-up development process. There is a lack of approaches (and corresponding implementations) that deal systematically with the whole development process. In contrast, approaches like [16] [17] focus mainly on early development activities as, e.g., requirement analysis or modeling, whereas approaches like [18] [19] provide basic implementation frameworks. Approaches like [20] [21] do provide a comprehensive development process but focus on self-adaptive systems based on centralized control concepts. An approach towards meta-adaptation support based on reusable and composable adaptation components is presented in [22]. The introduced *Transformer* framework constructs system global adaptation by contextually fusing adaptation plans from multiple adaptation components. Similar to the work presented in this paper, it focuses on decentralized structural adaptation for multiple purposes. While the work presented here focuses on a general engineering approach, the work presented in [22] focuses more on the conflict resolution between different adaptation behaviors.

In conclusion, it can be stated that there is a lack of approaches that combine the above mentioned aspects in order to support structural adaptivity as a basis for systematic development of generic self-organizing systems. Therefore, the following Section introduces an approach based on the systematical engineering of self-organizing MAS which supports the whole development process. It uses decentralized feedback structures and aims at supporting structural adaptivity in general.

III. SELF-ORGANIZING MAS

The presented approach on structural adaptation of coordinations processes is based on previous work on self-organizing dynamics in MAS. Such a self-organizing dynamic

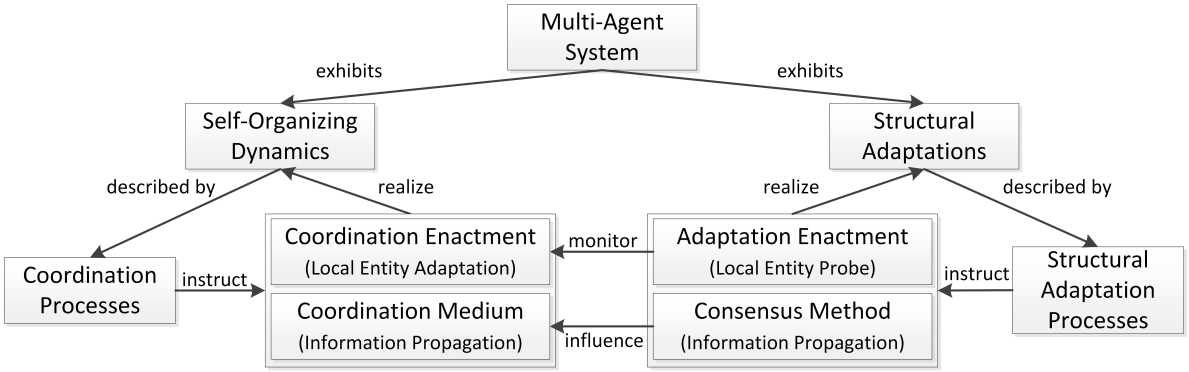

Figure 2. Structural adaptation processes exchange or reconfigure the coordination processes responsible for the self-organizing dynamics by monitoring SO performance indicators.

is shown in the left side of Figure 2. The MAS exhibits a self-organizing dynamic that causes the system to adapt to external and internal influences. The self-organizing dynamic realizes the intended adaptivity of the software system and is mapped by decentralized coordination processes. The processes describe a self-organizing behavior that continuously structures, adapts and regulates aspects of the application. They instruct sets of decentralized coordination media and coordination enactments. Coordination enactments and media distinguish between techniques for the adaptation of system elements (local entity adaptation) and realizations of agent interactions (information propagation). Together they control the microscopic activities of the agents, which on a macroscopic level lead to the manifestation of the intended self-organizing dynamic. The integration of the coordination enactments and media is prescribed by the coordination process definitions, which structure and instruct their operations. Thus the self-organizing dynamic of the MAS is described by coordination processes, which model the intended adaptivity of the system. On a technical level these processes instruct coordination enactments and media which realize the intended adaptivity.

Conceptually speaking, structural adaptations for self-organizing systems means an adaptation of the coordination processes, as they describe the system's intended self-organizing dynamic. The right side of Figure 2 illustrates this concept. The MAS exhibits structural adaptations which influence and observe the self-organizing dynamic. Similar to the self-organizing dynamic, the structural adaptations are described by processes. They define adaptation conditions, which specify states of the system where the self-organizing dynamic should be altered. This alteration is realized by prescribed adaptation activities, which are triggered due to specified SO performance indicators. Ineffective coordination processes are deactivated if the system's behavior becomes deficient and therefore other coordination processes are activated in exchange. This results in a structural adaptation of the coordination process composition. This means, on a lower level, if the system's behavior is not deficient but measured inefficient, the active coordination processes are reconfigured by parametric adaptations. The structural adaptation processes instruct adaptation enactments to monitor the agents with regard to the SO performance indicators. In both cases (structural or parametric adaptation), it is necessary that the system's entities find a consensus whether or not the adaptations should be performed. Therefore, a distributed consensus method is utilized in order to come to a decision about the execution of the adaptation. In case of a positive decision it is performed by manipulating the relevant coordination processes.

A. *Coordination Enactment Architecture*

The work on structural adaptations is based on a previously published tailored programming model for the software-technical utilization of coordination processes as reusable design elements [23]. The programming model provides a systematic modeling and configuration language called *MASDynamics* and a reference architecture to enable the enactment of pre-described coordination models called *DeCoMAS* [24] (Decentralized Coordination for Multi-Agent Systems). The architecture is based on the clean separation of activities that are relevant to the coordination of agents and the system's functionality. Therefore, coordination processes can be treated as first class design elements that define application-independent coordination interdependencies. Figure 3 illustrates the layered structure of the coordination enactment architecture. The functionality of the MAS is mapped by the application layer. The coordination logic is realized as an underlying layer. This layer provides a set of coordination media which provide the required coordination mechanisms. They build the infrastructure that allows the agents to exchange application independent coordination information and control the information dissemination. Thus, the coordination media are the technical realization of previous described coordination processes. The agents communicate with the coordination media using their coordination enactments (cf. Figure 3(B)). The enactments influence and observe the agent activities (1) and exchange information that is relevant for the coordination via the coordination media (2). The local configuration of these activities, e.g., when to publish information and how to process perceptions, is defined in a declarative, external coordination model (3) written in the *MASDynamics* language. Coordination is declaratively described to support the reuse of coordination pattern in different applications. This architecture focuses on the transparent separation of application and coordination logic, meaning that agent models are not modified by the coordination logic. This allows the supplement of coordination to existing applications. A recent example on how this architecture can be implemented to realize decentralized coordination in self-organizing systems based on Peer-to-Peer

technology is described in [25].

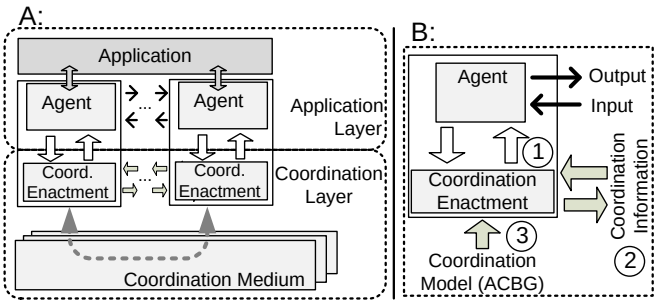

Figure 3. Coordination enactment architecture [26].

B. Engineering Self-Organizing MAS

The engineering of structural adaptations of coordination processes is part of an existing engineering approach for self-organizing MAS developed in the *SodekoVS* project [27]. Part of the project was the development of the *Coordination Enactment Architecture*. The project aims at providing self-organizing processes as reusable elements that developers can systematically integrate into their applications. The utilization of self-organization in software engineering is addressed by providing a reference architecture to offer a conceptual framework for the configuration and integration of self-organizing processes. The integration is guided by adjusting methodical development procedures. Following this, coordination media are made available as middleware services. A minimal intrusive programming model allows developers to configure and integrate representations of nature-inspired coordination strategies into their applications. Figure 4 denotes the conceptual view on integrating self-organization. Incremental development activities are supplemented with activities that address the manifestation of self-organizing phenomena (I-V). While developers design the functionality of their applications, they revise the decentralized coordination of component activities in interleaved development activities. Supplements to the requirements activities (I) facilitate the description of the intended application dynamics. During analysis activities (II) it is examined which instances or combinations of coordination metaphors can lead to the required adaptivity. Design activities (III) detail the models of selected coordination strategies and configure the coordination media that are used for their realization. These activities prepare the implementation and integration (IV) of medium instances to be configured and accessed by a generic usage interface. Testing (V) activities are supplemented with a simulation-based validation that agent coaction meets the given requirements, i.e., manifests the intended adaptiveness. The whole development process, as described in [27], is designed as a iterative process. Based on the results of the simulation-based validation, the self-organized coordination processes are redesigned until they achieve the intended adaptivity. Either based on the validation results or as an result of the initial analysis it may be observed that certain coordination processes or certain process configuration are only suitable for certain conditions but become deficient or insufficient for others. In this case it is practical to utilize structural adaptations for the re-composition of coordination processes or, on a lower level, parametric adaptations for the reconfiguration of coordination

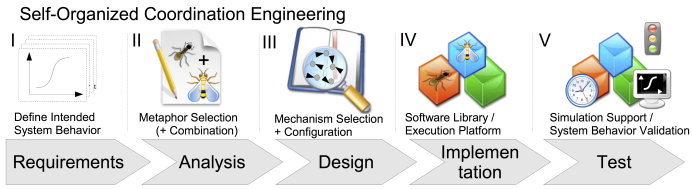

Figure 4. SodekoVS development activities following [27].

parameters. The key challenge hereby is to identify these conditions that require a structural (or parametric) adaptation and map them to SO performance indicators. As an addition to the existing engineering approach this paper propagates anticipated structural adaptations. Following the same iterative approach as designing and implementing the coordination processes, the adaptations should also be designed and implemented in a iterative way. The conditions that require adaptations should be identified based on the requirements and analysis activities and redefined by the results of a simulation-based validation.

C. Example: MarsWorld Coordination

This paper utilizes the MarsWorld scenario to explain the following concepts and their usage. The scenario is based on an application setting presented in [28]. A set of autonomous robots is sent to the planet of Mars to mine ore. The mining process consists of three distinct activities: (1) *analyze* locations to verify the presence of ore, (2) the mining or *production* of the ore and (3) *transporting* the mined ore to a homebase. The robots in this scenario are controlled by software agents, which are specialized to perform one of the distinct mining operations. The *Sentry* agents are equipped with sophisticated sensors to analyze potential ore locations, *Producer* agents have the capability to mine ore deposits at analyzed locations and the *Carry* agents can transport the mined ore to the homebase. Obviously as none of the three agents types is able to mine alone, the agents need to work together to achieve their collaborative goal. Therefore, this scenario is chosen for the case study, as the agents require some sort of coordination in order to achieve the collaborative goal.

The *Sentry* agents analyze potential ore deposits and inform the *Producer* agents whether or not ore can be mined there. The *Producers* mine the ore at the analyzed deposits and inform the *Carry* agents about it, so they can carry it to the homebase. Initially all agents explore the environment randomly. All agents have sensors to find potential ore deposits. If *Producer* or *Carry* agents encounter any potential deposits, they inform a *Sentry* agent. *Sentry* agents that have encountered a potential deposit or were informed about one, analyze the deposit. When they have verified the presence of ore at the location they request a *Producer* agent to mine the ore. Accordingly, after mining the ore *Producers* request a *Carry* agent to transport it to the homebase.

As the MarsWorld example exhibits no predefined organizational structure the agents need to be coordinated in order to achieve their collaborative goal. Based on the communication between the agents three coordination processes are needed to handle the information distribution:

1) Coordination information the *Producer* and *Carry* agents send to the *Sentry* agent, when they en-

countered potential ore deposits while exploring the environment.

2) Coordination information the *Sentry* agents send to *Producer* agents when they have analyzed a potential ore deposit and found ore to mine there.
3) Coordination information the *Producer* agents send to the *Carry* agents after ore was mined and is now ready for transportation to the home base.

As a simple example a coordination process manifestation based on a neighborhood approach is envisioned. In this case, each coordination process selects the agent of the appropriate type, which is nearest to the emitting agent. By selecting the nearest agent it is ensured that the informed agents have to travel the minimal distance to reach the designated destination. If the environment consists of multiple ore deposit clusters, characterized by a small distance between the ore deposits in the cluster and a large distance to the next cluster, this coordination approach lets the agents form local groups in a self-organized way. Thus, the different agent types will organize themselves in an emergent way.

Of course the coordination processes based on this approach require knowledge about the current positions of each agent. In the case study, the environment is equipped with a positioning service offering this information. The three coordination process realizations utilize the service in order to distribute the coordination messages to the nearest matching agent. This results in the realizations of the following three coordination processes:

- *latest_target_seen_nearest*: Whenever a *Producer* or *Carry* agent has encountered a potential ore deposit, this coordination process selects the *Sentry* agent nearest to the location of the *Producer* or *Carry* agent and informs it about the deposit. The *Sentry* agent adds the location to its queue and eventually analyzes it.
- *latest_target_analyzed_nearest*: After a *Sentry* has analyzed a potential ore deposit and actually found ore there, this coordination process selects the *Producer* agent nearest to the location and calls it to mine ore here.
- *latest_target_produced_nearest*: When a *Producer* agent has completely depleted the ore deposit, this coordination process selects and informs the *Carry* agent nearest to the location, so that it can transport the mined ore to the homebase.

Technical speaking, the coordination processes are described declaratively with the *MASDynamics* language. The *DeCoMAS* framework initializes the *Coordination Enactments* for the participating agents at application start time. These enactments monitor the agents and whenever one of the described *Coordination Events* occurs, the according *Coordination Information* is sent to the nearest matching agent. This is done by facilitating a *Coordination Medium*, which makes use of the environments positioning service to determine the nearest matching agent. The *Coordination Enactment* of the receiving agent triggers a *Coordination Event*, when the *Coordination Information* is received, letting the agent react based on its defined behavior.

IV. Structural Adaptation of Coordination Processes

This Section presents the structural adaptation architecture and the extension of the *MASDynamics* language for describing the adaptations in a declarative way. Also, an example for a structural adaptation based on the MarsWorld scenario is introduced.

A. Adaptation Architecture

The previous described Coordination Enactment architecture was extended by the introduction of so called *Adaptation Enactments*, to enable structural adaptations of coordination processes. Similar to Coordination Enactments, which were introduced to equip applications with coordination capabilities, the Adaptation Enactments equip applications with the capability to structural adapt coordination processes at runtime. Figure 5 shows the extension of the Coordination Enactment architecture. The Adaptation Enactments are part of the coordination layer and therefore independent from the system's functionality (supporting a clean separation between application and coordination logic). They observe the agents similar to the Coordination Enactments. But contrasting to the Coordination Enactments they do not influence the agents, but the coordination media, which are the technical realization of the coordination processes. As described before, the Coordination Enactments influence and observe the agent activities and exchange information relevant for coordination via a coordination medium. As shown in the Figure, the Adaptation Enactments consists of two components. The *Monitor* observes specified SO performance indicators as part of the agent, and in case of a decreased performance, determined by a specified condition, it initiates a consensus process for structural adaptations. The *Service* is used as an interface for the distributed consensus process. It offers generic consensus interface methods to support different consensus approaches (e.g., voting).

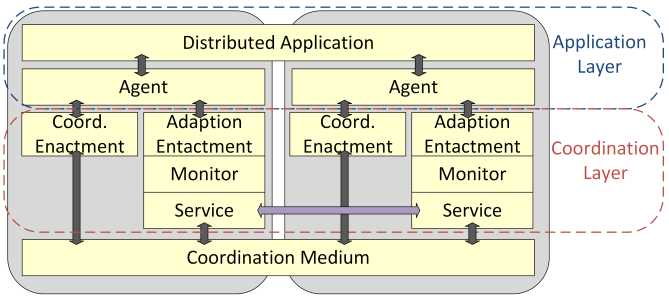

Figure 5. Adaptation architecture.

An example for a simple consensus algorithm is the following voting scheme. When the Adaptation Enactment Monitor of an agent observes a decreased performance, it initializes a distributed voting attempt and acts as leader of this vote. Utilizing the service interface it presents a suggested adaption to the other agents. Based on their local information (state of the agent), the called Adaptation Enactment Services decide whether or not to agree on the proposed adaptation and inform the vote leader about their decision. The leader analyzes the received votes and decides if the required majority for the

adaptation was reached. If so the suggested adaption is committed by activating or deactivating the affected coordination media, respective by changing their coordination parameters.

B. Adaptation Description

To describe structural adaptations the *MASDynamics* language was extended to support the declarative description of possible adaptations. These adaptations are described at design time and executed at runtime. As described before, structural adaptation of coordination processes can be realized by exchange or reconfiguration of coordination processes. Therefore, the already realized description of coordination processes was extended to indicate whether or not a coordination process is active at start time. This allows a developer to define multiple coordination processes with different behavior at design time, from whom only a subset may be active at start time. Furthermore, the *MASDynamics* language was extended with the following elements to describe these possible adaptations: The first part concerns the types of agents that are allowed to participate in the adaptation process. The *DeCoMAS* framework creates Adaptation Enactments for this types of agents, so they can be monitored with regards to specified performance indicators and are able to participate in the decision making process.

The second part concerns the actual adaptations. Each adaptation is identified by a unique *id* and a *reset* flag. The reset flags specifies if the performance indicator should be reset after a failed adaptation attempt. It can be used to prevent repeated adaptation attempts flooding the system, if only a subset of agents are subjected to bad performance indicators, while the majority of the systems still performs well and in that case would not agree on an adaptation. Further information in the adaptation description concerns the consensus process. It includes a minimum total number of *answers* a vote leader awaits, before it starts to evaluate the results from an adaptation attempt it started. Also the required *quorum* that has to be reached, before a structural adaptation may be accepted is specified. A *timeout* after which the vote leader will start to evaluate it, even if it has not received the required number of answers can also be specified. Furthermore, it is possible to *delay* an adaptation attempt if specified. Also, an adaptation can be blocked for a certain amount of time at start time, to avoid oscillation problems (*startDelay*).

Besides the information concerning the adaptation process, the description also contains information about the affected coordination processes. This includes the *realization id* of the affected coordination processes and the information if the process should be *activated* or *deactivated*, respective which *parameter* of the coordination process should be altered to which *value*. The last information needed for structural adaptations are the *constraints* regarding the agents state. For each agent type, they specify which *element* should be used as a SO performance indicator. They contain a *condition* and a *threshold*. The condition is used by the Adaptation Enactments Monitor, to point out if the performance indicator has become deficient for the specified agent and therefore, if an adaptation attempt should be started. When the Adaptation Enactment Service of an agent receives an adaptation request, it uses the threshold to determine whether or not it should agree to the proposed adaptation. Therefore, the threshold allows the specification of an insufficient performance indicator that is not as as strict as the actual condition, which would force the agent to start an adaptation attempt by itself. The threshold maps a negative trend allowing the agent to anticipate an insufficient performance.

C. Example: MarsWorld Structural Adaptation

In order to test the structural adaptations at runtime, the MarsWorld scenario was extended with an other manifestation of the three coordination processes described in Section III-C. The new manifestation is based on a simple random selection approach and therefore, does not exhibit any self-organizing behavior. In this case, each of the coordination processes randomly selects an agent of the appropriate type and informs it about the sensed, analyzed or produced ore.

Arguably, the coordination process manifestations that are based on the neighborhood approach will perform better, as the formation of local mining groups around the ore clusters minimizes the distance the agents have to travel, before they can analyze, produce or transport ore and therefore, optimizes the overall mining efficiency. But these coordination processes depend on the positioning service offered by the environment. If this service fails, the coordination processes will not be able to select the nearest agents, thus, they will not be able to inform the according agents. In such a case the application would benefit from the capability to structurally self-adapt and to switch to the random-selection based coordination process manifestations.

The goal is to deactivate the location based coordination processes, when the positioning service offered by the environment fails. As compensation the random selection based coordination processes will be activated. The agents have no knowledge about the fact that the positioning service has failed in this scenario conception. But they can measure the time that has passed since they have received the last coordination information. If they have not received any messages within a given time, they assume that something went wrong and the positioning service is broken. In this example, an agent waits 20 seconds until it assumes a malfunction and initializes a voting process acting as leader. Of course it is possible that agents have not received any coordination information within this time frame for other reasons. Therefore, they have to find a consensus, whether or not the coordination processes should be adjusted to overcome local phenomena. Agents that have received a voting requests determine if they have received any coordination information within the last 15 seconds (threshold) and if so vote *yes*. The agent, which started the voting process waits until it has received all the voting results (this is a simplification because in this small scenario we neglect any message lost) and evaluates them. If the required majority of 75% has been reached, the structural adaptation is performed. At start time, the adaptation capability is blocked for 30 seconds, because it may take a while before the first potential ore deposits are sensed and therefore, the system might adapt prematurely because of oscillation problems.

To measure the impact of the structural self-adaptation 50 simulations run with the location-based coordination processes active at start time were executed. After 60 seconds a failure in the environment's positioning service was simulated. Thus, the coordination processes were no longer able to select the nearest agent and therefore, were not able to distribute the coordination information. The adaptation capability was blocked for 30

 ISBN: 978-1-61208-391-9

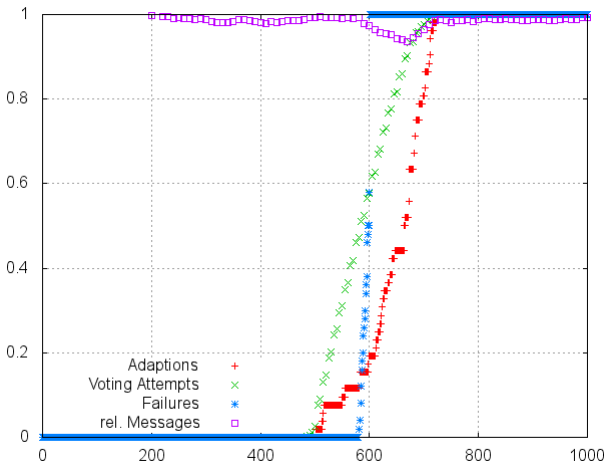

Figure 6. Evaluation of the Structural Adaptations for the MarsWorld (axis are described in the text).

seconds at start time, so 50 seconds had to pass before first voting attempts were started. Figure 6 shows the evaluation results of the self-adaptation. The x-axis denotes the time in 1/10 second steps. The y-axis shows the total number of voting attempts, the percentage of simulated failures in the positioning service and the percentage of adaptations. The Figure shows how the simulated failure in the location service occurred after 60 seconds have passed (deviations are caused by inaccuracies of the MAS platform's clock). It took 50 seconds before first voting attempts occurred. This is the shortest possible amount of time been passed before agents could observe the absence of any coordination information within the last 20 seconds. As Figure 6 shows, the percentage of voting attempts grows until all systems have performed the adaptation. In this case the adaptation was configured as *unique*, so after it was performed, no further voting attempts were undone.

The results also show that in approximately 50% of the simulation runs voting attempts occurred before the actual failure has arisen. This behavior can be explained by the random exploration of the environment and the communication cascades. Carry agents only receive coordination information after sentry agents have sensed ore and producer agents have mined it. If only a few ore deposits have been sensed after 50 seconds, the majority of the agents will not have received any coordination information until this time and therefore, interpret the absence of such messages as a potential failure. Thus, voting for an adaptation. A higher blocking time for the adaptation capability at start time or higher observation and threshold values would lead to fewer premature adaptations. On the other hand this would lead to a higher response time before the system adapts itself after a failure. As described before, suitable parameters have to be identified as part of the iterative, simulation driven engineering approach. The Figure also shows the relative number of coordination messages in relation to the number of coordination messages from a scenario were no failure occurs and therefore, no adaptation was needed. The curved line shows that there is no significant deviation between the two scenarios until the failure occurred. When the failure occurs the number of messages slides down in relation to the failure-free scenario. After the self-adaptation took place, the number of messages rises again to the level of the failure-free scenario. This shows how the self-adaption is able to repair a deficient behavior.

V. Conclusion and Future Work

In this paper, we presented an architecture and engineering approach for structural adaptations in self-organizing MAS to realize the dynamic exchange or reconfiguration of self-organizing coordination processes. It aims at supporting structural adaptations in general, rather than focusing on single self-* properties. The approach consists of a generic system architecture that governs the development of self-organizing MAS and a description language that supports the declarative description of coordination processes and pre-described structural adaptations, which can be processed by the corresponding framework automatically. It is supported by an engineering process consisting of incremental development activities that are supplemented with activities that address the manifestation of self-organizing behavior. The approach supports the modularization of coordination, which enables reusability and interoperability of coordination processes. It propagates a clear separation between application functionality and coordination, allowing developers to implement coordination without the need to change the applications business logic. Furthermore, with the introduction of structural adaptations of coordination processes it supports the self-adaptive structural exchange or reconfiguration of self-organizing processes. By detecting any decrease of specified SO performance indicators, the Adaptation Enactment extension initiates a distributed consensus process, that allows for the exchange or reconfiguration of the self-organizing processes to adapt to changing conditions automatically. Since the approach uses a set of coordination processes to be defined at design time, it is conceptually comparable to the reactive planning approach in a single BDI agent (local planing), but it strives towards finding a global structural adaptation plan for the system as a whole.

The presented framework was used in the *MarsWorld* scenario to realize a collaborative application in which three different types of agents needed to be coordinated, in order to mine ore on Mars. Therefore, three different coordination processes with two different manifestations (random or proximity-based) were implemented. The scenario was used as a proof of concept, to show how structural adaptations of coordination processes can be used. A specified SO performance indicator pointed out that an agent has not received any coordination messages within a given time, which led to an exchange of coordination processes at runtime.

Arguably, this work is still in progress and therefore, both the implemented distributed voting scheme as a consensus method and the MarsWorld scenario are kept quite simple and offer room for improvement. Future work will focus on these two aspects. A more sophisticated distributed consensus algorithm with a strong focus on decentralized coordination will be developed to replace the proof of concept voting scheme. Also the approach will be tested on more realistic and complex scenarios (e.g., the self-organized redistribution of bicycles in a bike sharing system [29]) involving different types of structural adaptations, in order to analyze which scenarios will profit from such adaptations and to improve the overall engineering approach.

 ISBN: 978-1-61208-391-9

ACKNOWLEDGMENT

The authors would like to thank Deutsche Forschungsgemeinschaft (DFG) for supporting this work through a research project on "Self-organization based on decentralized coordination in distributed systems" (SodekoVS).

REFERENCES

[1] K. Geihs, "Selbst-adaptive software," Informatik-Spektrum, vol. 31, 2008, pp. 133–145.

[2] A. S. Rao and M. P. George, "Bdi agents: From theory to practice," in Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 1995, pp. 312–319.

[3] W. Ren and R. W. Beard, Distributed consensus in multi-vehicle cooperative control : theory and applications, ser. Communications and control engineering. London: Springer, 2008.

[4] A. Vilenica, "Anwendungsentwicklung selbstorganisierender systeme: Systematische konstruktion und evaluierung," Ph.D. dissertation, Hamburg University, Department of Informatics, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, 12 2013.

[5] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, 2003, pp. 41–50.

[6] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck, "Organic computing - addressing complexity by controlled self-organization," in Proc. of the 2th Int. Symp. on Leveraging Appl. of Formal Methods, Verification and Validation, ser. ISOLA '06. IEEE Comp. Soc., 2006, pp. 185–191.

[7] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Software engineering for self-adaptive systems through feedback loops," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Engineering Self-Adaptive Systems through Feedback Loops, pp. 48–70.

[8] J.-P. Mano, C. Bourjot, G. A. Lopardo, and P. Glize, "Bio-inspired mechanisms for artificial self-organised systems," Informatica (Slovenia), vol. 30, no. 1, 2006, pp. 55–62.

[9] L. Veiga and P. Ferreira, "Poliper: policies for mobile and pervasive environments," in Proceedings of the 3rd workshop on Adaptive and reflective middleware, ser. ARM '04. New York, NY, USA: ACM, 2004, pp. 238–243.

[10] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM Trans. Auton. Adapt. Syst., vol. 4, May 2009, pp. 14:1–14:42.

[11] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli, "Case studies for self-organization in computer science," J. Syst. Archit., vol. 52, no. 8, 2006, pp. 443–460.

[12] T. D. Wolf and T. Holvoet, "A catalogue of decentralised coordination mechanisms for designing self-organising emergent applications," Dept. of Comp. Science, K.U. Leuven, Tech. Rep. CW 458, 8 2006.

[13] D. Garlan, "Model-based adaptation for self-healing systems," in In Proceedings of the first workshop on Self-healing systems. ACM Press, 2002, pp. 27–32.

[14] M. Smit and E. Stroulia, "Autonomic configuration adaptation based on simulation-generated state-transition models," in Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conf. on, 2011, pp. 175 –179.

[15] E. Yuan and S. Malek, "A taxonomy and survey of self-protecting software systems," in Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on, june 2012, pp. 109 –118.

[16] M. Morandini, F. Migeon, M.-P. Gleizes, C. Maurel, L. Penserini, and A. Perini, "A goal-oriented approach for modelling self-organising mas," in Proceedings of the 10th International Workshop on Engineering Societies in the Agents World X, ser. ESAW '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 33–48.

[17] T. De Wolf and T. Holvoet, "Towards a methodology for engineering self-organising emergent systems," in Proc. of the 2005 conference on Self-Organization and Autonomic Informatics (I). Amsterdam, The Netherlands: IOS Press, 2005, pp. 18–34.

[18] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky, "Metaself: an architecture and a development method for dependable self-* systems," in Proc. of the 2010 ACM Symp. on Applied Comp., ser. SAC '10. New York, NY: ACM, 2010, pp. 457–461.

[19] S.-W. Cheng, "Rainbow: cost-effective software architecture-based self-adaptation," Ph.D. dissertation, School of Computer Science, Carnegie Mellon Universit, Pittsburgh, PA, USA, May 2008.

[20] K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallenstein, G. Horn, M. Khan, A. Mamelli, G. Papadopoulos, N. Paspallis, R. Reichle, and E. Stav, "A comprehensive solution for application-level adaptation," Software: Practice and Experience, 2008.

[21] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. Papadopoulos, "A development framework and methodology for self-adapting applications in ubiquitous computing environments," Journal of Systems and Software, vol. 85, no. 12, Dec. 2012, pp. 2840–2859.

[22] N. Gui and V. De Florio, "Towards meta-adaptation support with reusable and composable adaptation components," in Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on, Sept 2012, pp. 49–58.

[23] J. Sudeikat and W. Renz, "MASDynamics: Toward systemic modeling of decentralized agent coordination," in Komm. in Vert.Syst. (KiVS). Springer, 2009, pp. 79–90.

[24] ——, "Decomas: An architecture for supplementing mas with systemic models of decentralized agent coordination," in IEEE/WIC/ACM International Conference on Web Intelligence. IEEE, 2009, pp. 104 – 107.

[25] T. Preisler, A. Vilenica, and W. Renz, "Decentralized coordination in self-organizing systems based on peer-to-peer coordination spaces," in Works. on Self-organising, adaptive, and context-sensitive distributed systems (SACS), vol. 56, 2013.

[26] A. Vilenica, J. Sudeikat, W. Lamersdorf, W. Renz, L. Braubach, and A. Pokahr, "Coordination in Multi-Agent Systems: A Declarative Approach using Coordination Spaces," in Proc. of the 20th European Meeting on Cybernetics and Systems Research (EMCSR 2010) - Int. Workshop From Agent Theory to Agent Impl. (AT2AI-7), R. Trappl, Ed. Austrian Soc. for Cyb. Studies, 4 2010, pp. 441–446.

[27] J. Sudeikat, L. Braubach, A. Pokahr, W. Renz, and W. Lamersdorf, "Systematically Engineering Self-Organizing Systems : The SodekoVS Approach," Electronic Communications of the EASST, vol. 17, 2009, p. 12.

[28] J. Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[29] T. Preisler, W. Renz, and A. Vilenica, "Bike-sharing system reliability problems - a data-based analysis and simulation architecture," in Sustainability and Collaboration in Supply Chain Management: A Comprehensive Insight into Current Management Approaches, ser. Supply Chain, Logistics and Operations Management, C. M. R. Wolfgang Kersten, Thorsten Blecker, Ed., vol. 16. Josef Eul Verlag, 8 2013, pp. 83–97.

 ISBN: 978-1-61208-391-9