Structural Adaptation for Self-Organizing Multi-Agent Systems: Engineering and Evaluation

Thomas Preisler, Wolfgang Renz, and Tim Dethlefs

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

***Abstract*—Over one decade of research in engineering of self-organization (SO) has established SO as a decentralized way to develop self-adaptive systems. However, such SO systems may show unwanted behavior under certain conditions, e.g., a decrease of performance and/or starvation, even when apparently well-engineering. 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 adaptation process that allows the exchange (or reconfiguration) of the self-organizing coordination processes, enabling the system to adapt to changing conditions automatically. The proposed engineering approach is explained exemplary and evaluated using a case study in which a set of autonomous robots is needed to be coordinated in order to achieve their collaborative goal of mining commodities. Based on two different coordination strategies, it will be shown how the autonomous entities benefit from self-organizing coordination processes that support the emergent formation of local mining groups among the robots, and how the concept of structural adaptation helps to overcome a failure that negatively influences these coordination processes by a dynamic exchange of the coordination processes.**

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

I. INTRODUCTION

A preliminary version of this article appeared as a conference paper [1]. This article extends the previous conference paper by giving a more detailed description of the both, the engineering approach as well as the evaluation of the proposed structural adaptations for self-organizing Multi-Agent Systems (MAS).

For self-organizing 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 at such adaptive systems, they reveal many different facets. According to [2] 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 distinguishes between approaches where the self-organizing behavior is achieved through centralized or decentralized control concepts. 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. Our approach is based on a decentralized architecture and emphasizes structure-based changes (while also supporting parameter-based ones). Possible adaptations are modeled manually at *design-time* and executes automatically at *run-time*. Therefore, the solution is ranked between manually and automatically adaptations.

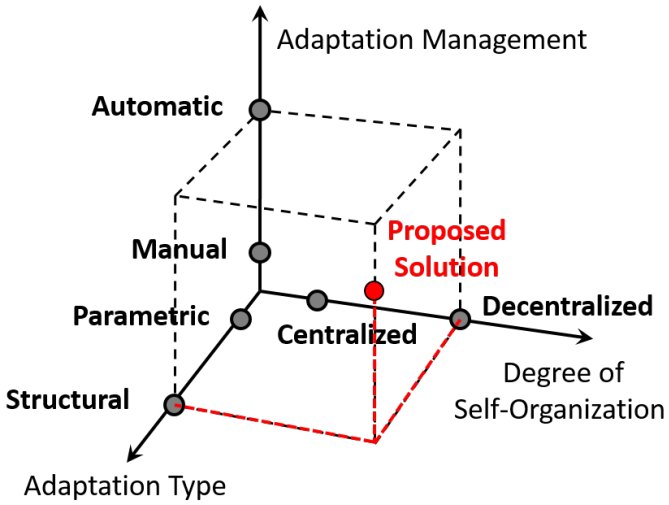

Figure 1. Dimensions of adaptive systems following [3].

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 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 mechanisms. It is conceptual comparable to the local reactive planning approach from the BDI (belief, desire, intention) agent software model [4], where an agent selects a plan based on local information and alternatives. In contrast, the proposed approach deals with systems that strive towards a distributed consensus on an application-global level to select and execute predefined structural adaptation plans. Thereby, each of the participating agents only relies on its local information in a decentralized way.

The remainder of this paper is structured as follows: Section II describes related work, followed by Section III where the properties and engineering challenges of self-organizing MAS are described. Section IV describes the core concepts of structural adaptations, the proposed architecture and description language as well as the adaptation algorithm. Section V introduces the *MarsWorld* scenario and describes two different coordination process manifestations as well as whose structural adaptation. The evaluation results of both the differences between the two coordination process manifestations and the impact of the structural adaptation are described in Section VI. Finally, Section VII 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 engineering and operating approaches for distributed systems substantially. Years ago, such systems were mainly closed ones with a-priori known tasks, challenges, and users. Nowadays, with an increasing pervasiveness and distribution, they have turned into an integral part of the business world as well as the private life of many people. This evolution implicates new challenges for the engineering and operation of distributed systems. For instance, 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 exhibit adaptivity. Here, feedback loops consist of three main components: *sensor*, *actuator* 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 that serves as a connector between the system input (sensor) and the output (actuator) can be very different with regards to 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, the authors of [9] introduce 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 described in this paper, the work presented in [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, which 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 mentioned in e.g., [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. However, this 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 that 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 with the whole development process systematically. Approaches like [16] [17] focus mainly on early development

activities (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. The authors of [23] introduce a decentralized planning approach for self-adaptation in multi-cloud environments. Thereby, they focus on the runtime management of Internet of Things oriented application deployed in multi-clouds. Decentralized self-adaptation is seen as a promising solution in order to maintain the applications for quality assurance. The presented approach tackles the uncertainty effect of adaptations on a specific layer, which may cause negative impacts on other layers. Therefore, they introduce a planning model and method to enable the decentralized decision making. The planning is formulated as a Reinforcement Learning problem [24] and solved using the Q-learning algorithm [25]. Contrasting, to the approach presented here where a general approach for the engineering of self-adaptation is envisioned, the approach presented by [23] focuses on solving the problem of self-adaptation in the specific context of multi-cloud environments.

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 two sections introduce 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 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 mean 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 self-organization 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, that 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 self-organization 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 voting method is utilized in order to find an agreement about the execution of the adaptation. In case of a positive one 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 [26]. The programming model provides a systematic modeling and configuration language called *MAS-Dynamics* and a reference architecture to enable the enactment of pre-described coordination models called *DeCoMAS* [27] (Decentralized Coordination for Multi-Agent Systems). The architecture is based on the clean separation of activities that are relevant for 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 the 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

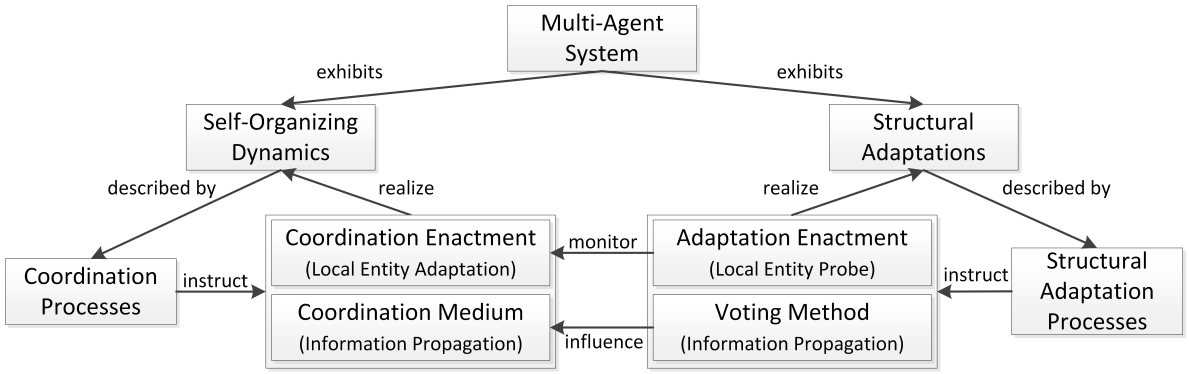

Figure 2. Structural adaptation processes exchange or reconfigure the coordination processes responsible for the self-organizing dynamics by monitoring self-organizational performance indicators.

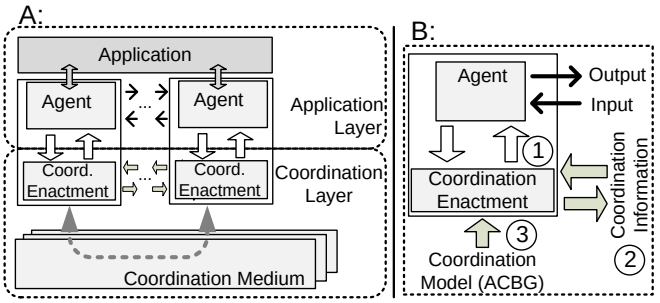

Figure 3. Coordination enactment architecture [29].

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 [28].

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 [30]. Part of the project was the development of the *Coordination Enactment Architecture*. The project aimed 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 [30], 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 suitable for some conditions but become deficient or insufficient for others. In this case it is practical to utilize structural adaptations for the recomposition of coordination processes or, on a lower level, parametric adaptations for the reconfiguration of coordination parameters. The key challenge hereby is to identify these conditions that require a structural (or parametric) adaptation and map them to self-organizational 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 an 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.

IV. Structural Adaptation of Coordination Processes

This section presents the structural adaptation architecture and an extension of the *MASDynamics* language for the declarative description of adaptations. The proposed architecture

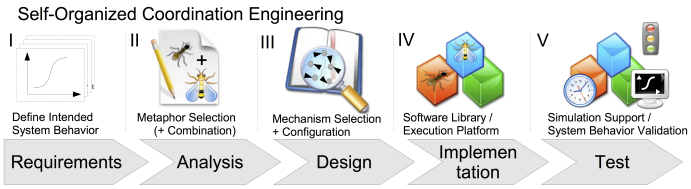

Figure 4. SodekoVS development activities following [30].

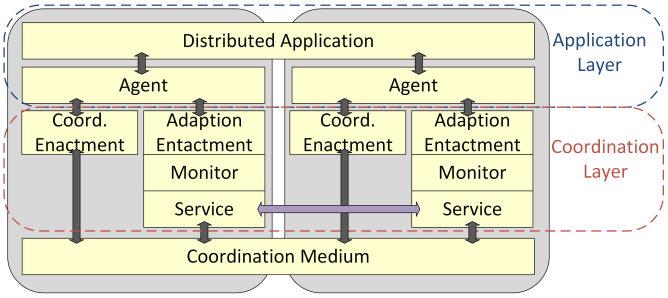

Figure 5. Structural Adaptation Architecture.

facilitates a voting algorithm that is used to find an agreement on possible adaptations which are then executed.

A. Adaptation Architecture

The previous described Coordination Enactment Architecture is extended by the introduction of so called *Adaptation Enactments* in order to enable structural adaptations of coordination processes. Similar to the Coordination Enactments, that 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 self-organizing performance indicators, which are part of the agent, and in case of a decreased performance, determined by a specified condition, it initiates a voting process for a predefined structural adaptation. The *Service* is used as an interface for the distributed voting process. 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 suggests an adaptation 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 adaptation is committed by activating or deactivating the affected coordination media, respectively by changing their coordination parameters.

B. Adaptation Description

In order 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 adaptations 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 developers to define multiple coordination processes with different behaviors 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 an 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 voting 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 *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 the received answers even if it has not received all of the required 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 process 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 self-organizing performance indicator. They contain a *condition* and a *threshold*. The condition is used by the Adaptation Enactment 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. Listing 1 shows the XML Schema Definition (XSD) for the adaptation part of the *MASDynamics* language.

Listing 1. XSD for the adaptation part of the *MASDynamics* language.

```
<xs:schema attributeFormDefault="unqualified"
    elementFormDefault="qualified" xmlns:xs="http://
    www.w3.org/2001/XMLSchema">
  <xs:element name="adaptation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="realizations">
          <xs:complexType>
            <xs:sequence>
    <xs:element name="realization" maxOccurs="unbounded
        " minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
 <xs:attribute type="xs:string" name="id" use="optional
     "/>
 <xs:attribute type="xs:string" name="activate" use="
     optional"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="constraints">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constraint" maxOccurs="
                  unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
 <xs:attribute type="xs:string" name="agent_id" use="
     optional"/>
 <xs:attribute type="xs:string" name="element" use="
     optional"/>
 <xs:attribute type="xs:string" name="type" use="
     optional"/>
 <xs:attribute type="xs:byte" name="condition" use="
     optional"/>
 <xs:attribute type="xs:byte" name="threshold" use="
     optional"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:string" name="id"/>
      <xs:attribute type="xs:byte" name="answers"/>
      <xs:attribute type="xs:float" name="quorum"/>
      <xs:attribute type="xs:short" name="startDelay"/>
      <xs:attribute type="xs:string" name="reset"/>
      <xs:attribute type="xs:string" name="single"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

C. Adaptation and Voting Algorithm

The adaptation and voting process is illustrated as an UML Activity Diagram in Figure 6. The whole process is started when a self-organization performance indicator changes its value. Whenever such an event occurs the adaptation enactment's monitor checks the adaptation constraints. If a constraint is fulfilled and no adaptation process with a higher priority is currently running, the according adaptation process is started. The priority of an adaptation process is determined by the hash value of the initiating agent's identifier. Due to the low probability of collisions in hash values, they can be used as unique identifiers in distributed systems where no global knowledge about all participating entities exists. By checking for an adaptation processes with a higher priority it is guaranteed that only one adaptation process is executed simultaneously. Therefore, interferences between these processes are prevented. If an adaptation constraint is fulfilled and no adaptation process with a higher priority is currently running, the monitor will call the remote voting services of all other agents. Then it waits until the required number of answers, as specified in the adaptation description, is received or until the timeout, also specified in the description, occurs.

A voting service that receives a voting request will check if it had already been suggested another adaptation with a higher priority, again to avoid inferences, in such a case it votes *no*. Otherwise, it checks its local value of the specified performance indicator against the threshold value. If the threshold is reached, a negative performance trend is recognized and the voting service agrees on the suggested adaptation by voting *yes*.

The monitor that proposed the adaptation waits until it has received the required number of answers or the timeout has occurred. Then it checks if the required majority, as specified in the adaptation description, was reached. The adaptation process is aborted in case of a timeout. When the monitor has received the required number of answers it evaluates them by counting the *yes* and *no* votes and calculating the ratio. If the ratio reaches or exceeds the required majority, the monitor commits the suggested adaptation. If not, it aborts the process. Again if adaptation process was interfered by a process with a higher priority it is aborted at this point. The proposed adaptation is committed by activating, respectively deactivating the distributed coordination by or by changing their configuration parameters via interfaces offered by the coordination media for such purposes.

V. Case Study: MarsWorld

The MarsWorld scenario is based on a hypothetical application setting presented in [31]. A set of autonomous robots is sent to the planet Mars to mine ore. The mining process consists of three distinct activities:

1) *Analysis* of potential ore locations to verify the presence of ore.
2) Mining or *production* of the analyzed ore deposits.
3) *Transportation* of the mined ore to a homebase.

The robots in this scenario are controlled by software agents, which are specialized to perform one of the three 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 their collaborative goal. The *Sentry* agents

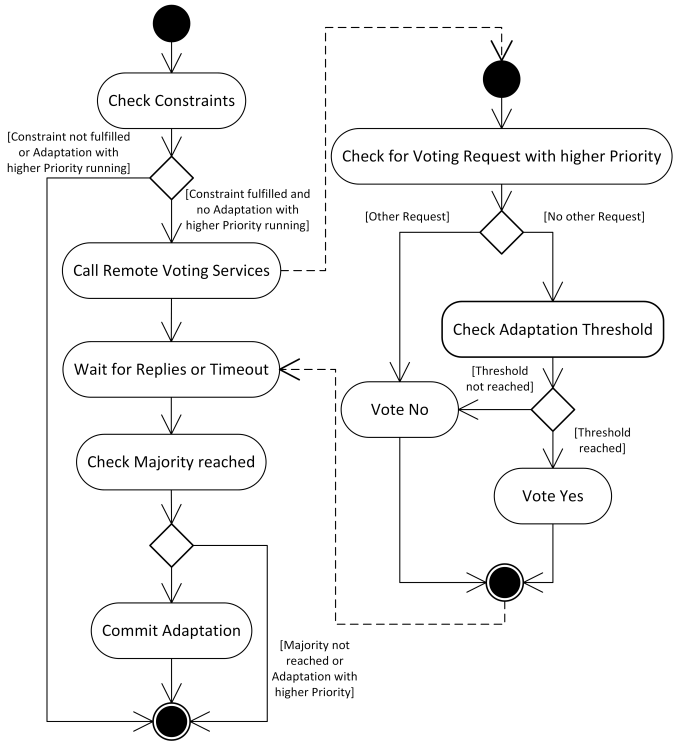

Figure 6. Adaptation and Voting Algorithm.

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 are equipped with 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, move towards the deposit and analyze it. When they have verified the presence of ore at the location they request a *Producer* agent to mine it. Accordingly, after mining the ore *Producers* request a *Carry* agent to transport it to the homebase.

A. Coordination

As the MarsWorld example exhibits no predefined organizational structure the agents need to be coordinated in order to achieve their collaborative goal. Three coordination process for the distribution of the required information can be derived from the described communication taking place among the agents:

1) Coordination information the *Producer* and *Carry* agents send to the *Sentry* agent, when they encounter 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 they minded ore and it is now ready for transportation to the home base.

In order evaluate the impact of both, the self-organizing coordination processes and their structural adaptation two different manifestations of the three described coordination processes are envisioned. The first one 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 required type and informs it about the sensed, analyzed or produced ore.

The second manifestation is based on a neighborhood approach. In this case, each coordination process selects the agent of the required 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 deposits in the cluster and a large distance to the next cluster, this coordination approach leads to the self-organized formation of local mining groups among the agents. Thus, enabling the agents to organizes themselves in an emergent way.

Obviously, knowledge about the current positions of each agent is required by the coordination processes based on this approach. Therefore, the environment in this case study is equipped with a positioning service offering this information. The three coordination process realizations based on the neighborhood approach utilize this service in order to distribute the coordination messages to the nearest agent of the appropriate type. This results in the realizations of the following six coordination processes:

- *latest_target_seen_random*: This coordination process informs a *Sentry* agent whenever a *Producer* or *Carry* agent has encountered a potential ore deposit. It randomly selects one of the *Sentry* agents and informs it. The *Sentry* agent adds the received location to its queue of locations which it has to analyze.
- *latest_target_analyzed_random*: This coordination process randomly selects and informs one of the *Producer* agents if a *Sentry* agent has analyzed a potential ore deposit and actually found ore to mine there. The *Producer* agents adds the location to its queue of ore deposits to mine and eventually starts to mine it until the deposit is depleted.
- *latest_target_produced_random*: This coordination process randomly selects and informs one of the *Carry* agents whenever a *Producer* agent has completely mined a ore deposit. The *Carry* agent adds the location to its queue of produced ore that is ready for transportation
- *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.

Listing 2 shows the declarative description of the *latest_target_seen_nearest* coordination process. It is written in the previously described *MASDynamics* configuration language. The listing shows that whenever the *callSentryEvent* is monitored in either a *Producer* or *Carry* agent, coordination information about this latest seen target is communicated to a *Sentry* agent. The according event is triggered inside a *Producer* or *Carry* agent whenever a potential ore deposit is found. Using the specified coordination mechanism, which is the technical realization of the previous described coordination medium concept, the nearest *Sentry* agent is selected as the receiver of the coordination information. The coordination mechanism is implemented as Java class and makes use of the positioning service offered by the environment in order to determine the nearest *Sentry* agent. When the *Sentry* agent receives the coordination information with the position of the potential ore deposit, the specified *latestTargetEvent* is triggered by the its coordination enactment. This results in the location being added to the *Sentry*'s queue of location to analyze. The coordination process descriptions for the other coordination processes are constructed similar.

Listing 2. XML listing showing the declarative description of the coordination process realization for the *latest_target_seen_nearest* process written in the *MASDynamics* language.

```
<realization id="latest_target_seen_nearest">
 <from>
  <agent_element element="callSentryEvent"
       agent_id="Producer" type="INTERNAL_EVENT">
   <parameter_mappings>
    <mapping ref="latest_target" name="
         latest_target"/>
   </parameter_mappings>
  </agent_element>
  <agent_element element="callSentryEvent"
       agent_id="Carry" type="INTERNAL_EVENT">
   <parameter_mappings>
    <mapping ref="latest_target" name="
         latest_target"/>
   </parameter_mappings>
  </agent_element>
 </from>
 <mechanism_configuration
  mechanism_id="sodekovs.marsworld.coordination.
       NearestMechanism" agent_type="JADEX">
 </mechanism_configuration>
 <to>
   <agent_element element="latestTargetEvent"
        agent_id="Sentry" type="INTERNAL_EVENT">
   <parameter_mappings>
    <mapping ref="latest_target" name="
         latest_target"/>
   </parameter_mappings>
  </agent_element>
 </to>
 <active>true</active>
</realization>
```

B. Structural Adaptation

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. Therefore, it 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 the agents have not received any coordination information within this time frame for other reasons, e.g., when the agents are exploring an area of the scenario where no other agents are exploring and, therefore, these agents are to far away from the emitting agents to be selected by the location based coordination processes. Therefore, the agents have to find an agreement, whether or not the coordination processes should be adjusted to overcome local phenomena.

Therefore, the voting algorithm described in Section IV-C is used by the agents to find an agreement on the adaptation. When an agent has not received any coordination information within the last 20 seconds (condition of the adaptation constraint) it starts the voting process and acts as the leader of it. Agents that have received a voting requests determine if they have received any coordination information within the last 15 seconds (adaptation threshold) and if so vote *yes*.

The threshold value expresses tendencies indicating a potential negative trend, so that agents can agree on a proposed adaptation before exhibiting a deficient behavior by themselves. In this case, they exhibit a potential negative trend and interpret it as an upcoming deficiency if an other agent proposes an adaptation and therefore, already has exhibited the deficiency. Of course, more complex conditions, as the time windows used in this example, are possible to model both the constraint (deficient behavior) and threshold (negative trend) values.

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.

Listing 3 shows the declarative description of the structural adaptation for this example. It shows that the agent initiating the voting process awaits answers from 16 others (including itself) and that a quorum of 75% *yes*-Votes is required so that the adaptation is performed. The *startDelay* indicates that the adaptation enactment is blocked for the first 30 seconds. The *reset* flag appoints that the constraint value should not be reseted after a failed voting attempt and the *single* flag

asserts that the adaptation should only be performed once. Under the *realizations* tag it is described which coordination processes should be activated and deactivated if the adaptation is performed. The *constraints* describe which values in the participating agents should be monitored, in this case an BDI[1]-agent belief called *no_msg_received* that stores the information how much time has passed since the agent has received the last coordination information. As described before, the *condition* and *threshold* values are simple integer values in this example.

Listing 3. XML listing showing the declarative description of the structural adaptation for the MarsWorld example.

```
<adaptations>
 <adaptation id="change-to-random" answers="16"
      quorum="0.75" startDelay="30000" reset="
      false" single="true">
  <realizations>
   <realization id="latest_target_seen_nearest"
        activate="false"/>
   <realization id="latest_target_analyzed_nearest
        " activate="false"/>
   <realization id="latest_target_produced_nearest
        " activate="false"/>
   <realization id="latest_target_seen_random"
        activate="true"/>
   <realization id="latest_target_analyzed_random"
        activate="true"/>
   <realization id="latest_target_produced_random"
        activate="true"/>
  </realizations>
  <constraints>
   <constraint agent_id="Producer" element="
        no_msg_received" type="BDI_BELIEF"
        condition="20" threshold="15"/>
   <constraint agent_id="Carry" element="
        no_msg_received" type="BDI_BELIEF"
        condition="20" threshold="15"/>
   <constraint agent_id="Sentry" element="
        no_msg_received" type="BDI_BELIEF"
        condition="20" threshold="15"/>
  </constraints>
 </adaptation>
</adaptations>
```

C. Scenario Description

The following scenario was designed to measure and evaluate both, the differences between the self-organizing and the random-selection based coordination processes and the impact of the structural adaptation. Figure 7 shows the examined scenario. It is characterized by a *2 x 2 dimension* and contains *20 ore deposits* with a total ore amount of *1000 ore units*. As shown in the figure, the ore deposits are grouped together to local clusters. Each cluster contains *250 units of ore*. The ore deposits in the left, top corner of the scenario are identified as cluster 1, the ore deposits in the right, top corner as cluster 2, the ore deposits in the left, bottom core as cluster 3 and the ore deposits in the right, bottom corner as cluster 4. The scenario was designed this way to highlight the self-organizing potential of the neighborhood-based coordination approach, allowing the agents to form local mining groups in a self-organized way. The homebase is located in the middle of the environment.

At the start of the application, all agents start from the homebase and begin to explore the environment randomly. When the *Sentry* agent encounters any ore deposit it starts to analyze it, while the *Producer* and *Carry* agents inform *Sentry*

[1]The belief-desire-intention agent model is developed for programming intelligent agents. Superficially characterized by the implementation of an agent's beliefs, desires and intentions, it uses these concepts to solve a particular problem in agent programming.

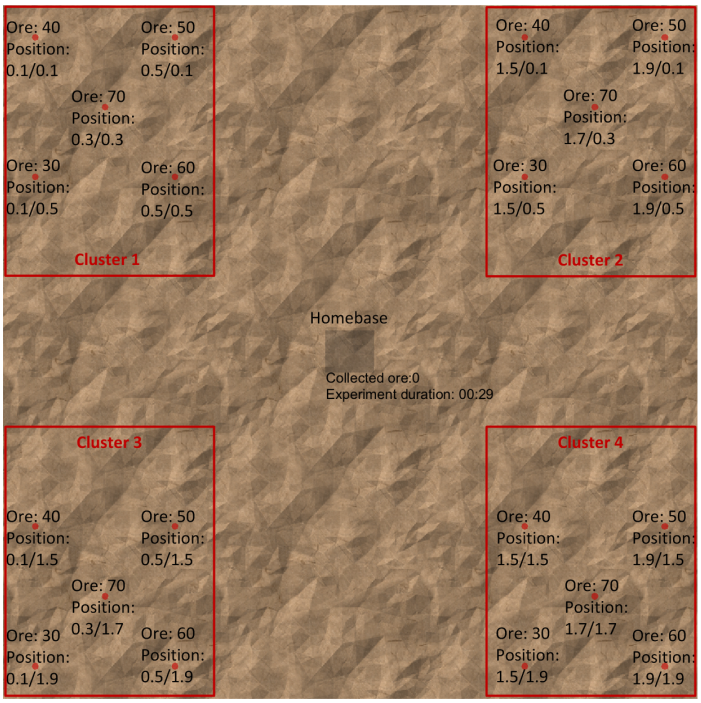

Figure 7. Screenshot of the MarsWorld scenario with highlighted ore clusters.

agents, based on the used coordination process, about sensed deposit. The scenario is finished when the complete amount of ore was mined and transported to the homebase. It is executed with the following amount and configuration of agents:

- *Sentry*: Amount 4, Vision 0.15, Speed 0.1
- *Producer*: Amount 4, Vision 0.1, Speed 0.1
- *Carry*: Amount 8, Vision 0.05, Speed 0.2, Capacity[2] 20

Figure 8 shows a running MarsWorld application based on this scenario. As shown in the figure, the agents were able to mine 150 units of ore after 59 seconds have passed. The shown scenario was executed with the neighborhood-based coordination process manifestations and at this point it seems that the agents have formed three local mining groups (black markers) while mining the ore clusters two, three and four.

VI. Evaluation

The realization of the MarsWorld scenario is based on the Jadex MAS platform [32]. The *Sentry*, *Producer* and *Carry* agents were realized as BDI-agents. At start time, all agents explore the environment randomly. Based on the coordination information they receive through to the coordination processes goals to analyze (*Sentry*), produce (*Producer*) or transport (*Carry*) ore are dispatched in the according agents.

In order to measure and evaluate the impact of both, the self-organizing coordination processes and the structural adaptation of these processes, the *MarsWorld* scenario was executed 50 times for each of the two different coordination process manifestations (neighborhood-based and random-selection) as

[2]The capacity denotes the maximum number of ore units a *Carry* agent can transport.

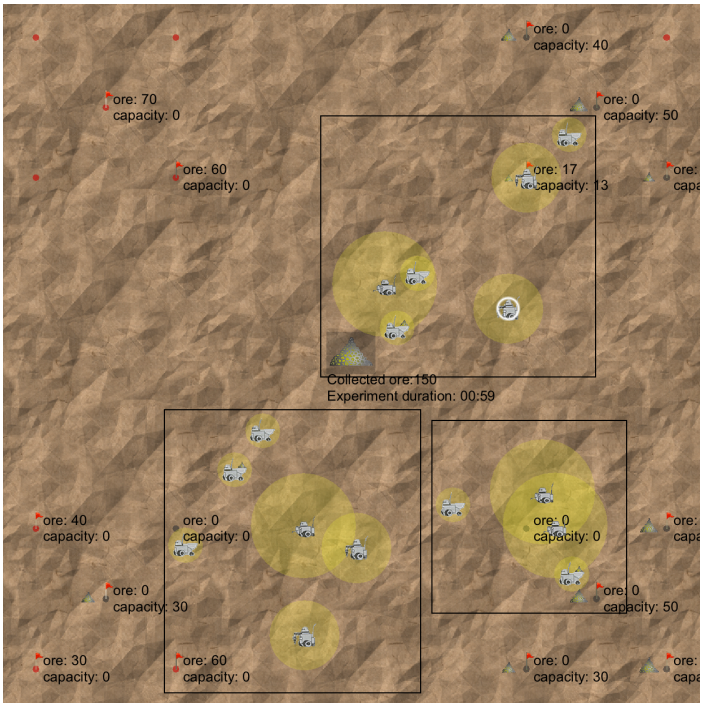

Figure 8. Screenshot of the running MarsWorld application with highlighted self-organizationally formed local groups.

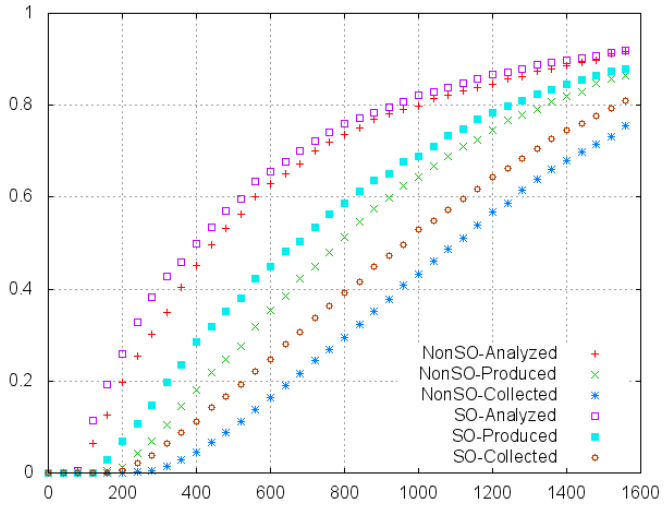

Figure 9. Evaluation of the coordination processes.

well as for the structural adaptation. To automatically start the 50 simulation runs for each of the three described characteristics, a simulation management framework for agent based distributed systems [33] was used. Additionally, an evaluation framework for the automatically observation and evaluation of predefined system observables [34] was used.

The six different coordination processes were described declaratively using the *MASDynamics* language. As a technical realization two different coordination mechanisms were implemented. The first one randomly selects one agent of the appropriate type. This coordination mechanism was used by the three random-selection based coordination process manifestations. The other coordination mechanism used the positioning service offered by the environment to select and inform the nearest agent of the appropriate type. This coordination mechanism was used by the three neighborhood based coordination process manifestations.

A. Coordination

In order to evaluate the impact of the self-organization, the number of analyzed, produced and collected ore was observed in each of the 50 simulation runs for the neighborhood and random-selection based coordination process manifestations. Figure 9 shows the results of the analysis. The time passed is denoted on the x-axis in 1/10 seconds steps. The y-axis shows the percentage amount of ore that is analyzed, produced or collected. As described in Section V-C, the scenarios had a total amount of 1000 units of ore. Only the first 3 minutes of all simulation runs were observed, because this was the fastest time in which the scenario was completely depleted in all of the simulation runs and, therefore, the only period under observation that contains results for all simulation runs. The results of the neighborhood-based coordination process manifestations are identified by the *SO*-prefix in Figure 9 and the ones based on the random-selection coordination process manifestations by the *NonSO*-prefix.

The efficiency of the self-organization is shown in Figure 10. Again the x-axis denotes the passed time in 1/10 second steps and the y-axis shows the percentage efficiency of the self-organizing coordination processes over the non self-organizing ones. It shows how the self-organized coordination processes which enable the agents to form local mining groups by calling the nearest available agent of the required type increase the overall efficiency. On a more detailed view, the figure shows that the impact of the self-organization on the amount of analyzed ore is significantly less then the impact on the amount of produced and collected ore. This is because the *Sentry* agents are able to find potential ore deposits to analyze by themselves while they are exploring the environment randomly. Therefore, they do not necessarily rely on receiving coordination information about potential ore deposits. These information just helps the agents to find deposits faster. The *Producer* and *Carry* agents, on the other hand are only able to produce or carry ore when they have received a coordination information about an analyzed or mined deposit. Thus, their ability to produce or carry ore strongly depends on the coordination. Contrary, the *Sentry* agents are able to analyze ore without any coordination. Furthermore, the figure shows how the efficiency of the three self-organized coordination processes differs over time depending on the amount of ore that has already been analyzed, produced or carried.

B. Structural Adaptation

To measure the impact of the structural self-adaptation 50 simulation run were executed with the self-organizing coordination processes active at start time. After 60 seconds a failure in the positioning service of the environment was simulated. Thus, the self-organizing coordination processes relying on this service were no longer able to select the nearest appropriate agent and therefore, were not able to distribute coordination information anymore. As described before, the Adaptation Enactment was blocked for the first 30 seconds

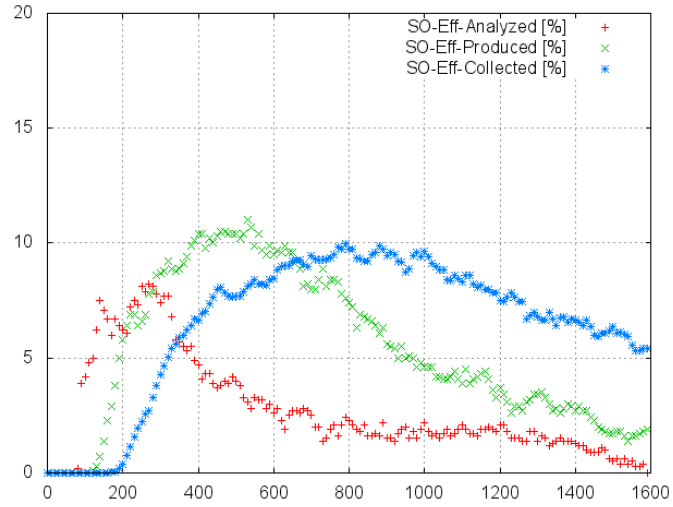

Figure 10. Efficiency of the self-organized coordination processes.

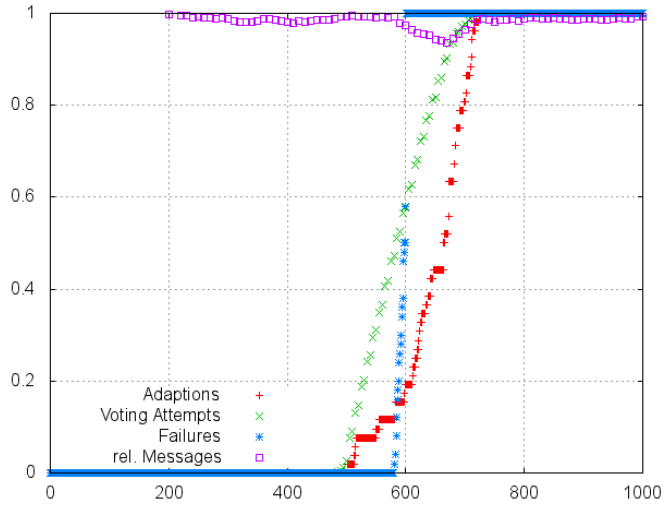

Figure 11. Evaluation of the structural adaptation.

to overcome problems due to agents still exploring the environment without having sensed any potential ore deposits and therefore, not sending any coordination information. Figure 11 shows the evaluation results of the structural adaptation. Again 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 performed adaptations.

The figure shows how the simulated failure in the positioning service occurs after 60 seconds have passed (deviations are coursed by inaccuracies of the MAS platform's clock service). First voting attempts can be observed after 50 seconds have passed, as this is the shortest possible time after which agents could notice not having received any coordination information within the last 20 seconds. As the figure shows, the percentage of voting attempts grows until in mean all systems have performed the structural adaptation. In this case the adaptation was configured to be an *unique* adaptation. So, after it was performed, no further voting attempts were started. The results also show that in approximately 15% of the simulation runs, the system adapted its structure before the actual failure occurred. As described before, the adaptation requires a majority of 75% percent of the agents agreeing on it. As the scenario consists out of 16 agents, 12 agents are required to vote for the adaptation. For the voting attempt to be started, at least one agent must not have received any coordination information within the last 20 seconds. In order to reach the required majority, this requires at least 11 agents not having received any coordination information within the last 15 seconds (the specified threshold value). Due to the random exploration of the environment this phenomena might occur before the simulated failure, if the agents have only sensed a few ore deposits. Combined with the formation of local self-organized mining groups, this can lead to a majority of agents that are not involved in any mining activities and, therefore, are not receiving any coordination information. These agents will interpret the absent of such messages as a potential failure in the coordination processes and thus, vote for the structural adaptation. A higher blocking time for the adaptation enactment at start time or higher constraint and threshold values would lead to fewer premature adaptations. But on the other hand, that would also lead to a higher response time before the system adapts itself after a failure. As described in Section III-B, suitable adaptation parameters have to be identified as part of an iterative, simulation driven engineering approach.

Furthermore, the figure also shows the relative number of coordination messages from the defective scenarios, in relation to the number of coordination messages from the self-organized scenarios where no failure occurs and therefore, no adaptation was needed. The curved line shows that there is no deviation between the two scenarios until the failure occurs. When the failure occurs the number of messages slides down in relation to the failure-free scenario. After the structural adaptation took place, the number of messages rises up until the level of the failure-free scenario is reached again. This shows how the structural adaptation is able to repair a deficient behavior caused by an external failure.

VII. 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 application's 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 voting process, that allows for the exchange or reconfiguration of the self-organizing processes to adapt to changing conditions automatically. It is conceptual comparable to the reactive planning approach (local) from the BDI agent model, where an agent selects a plan based on local information and alternatives. In case of the proposed approach, the system as a whole strives towards a distributed consensus on application-global level to select and execute predefined structural adaptation plans. Thereby, each of the participating agents only relies on its local information.

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. This scenario was used to evaluate both the impact of self-organizing coordination processes and their structural adaptation. In order to show the impact of self-organizing behavior three different coordination processes in two different manifestations were implemented. A random-selection based manifestation and a proximity-based one. The later one allows the agents to form local mining groups in a self-organized way by exchanging coordination information with the nearest possible agent. For each of the two coordination process manifestations 50 simulation runs were performed and the number of analyzed, produced and collected ore was measured. Based on these results, it was shown that the application benefited from self-organizational behavior, as the self-organizing proximity-based coordination processes were more efficient in terms of processed ore over time than their non self-organizing counterparts. The impact of structural adaptations was shown by simulating a failure in the positioning service offered by the environment and used by the proximity-based coordination processes. The agents' Adaptation Enactments detected and interpreted the absence of coordination messages caused by the failure of the positioning service as a deficient behavior and agreed on a structural adaptation using a distributed voting approach. The deficient proximity-based coordination processes were deactivated and replaced by their random-selection based counterparts. Thus, we showed how the system was to able to recognize the failure of the positioning service, by observing the absence of incoming coordination information and adapting it's structure to repair itself. Furthermore, it was explained under which conditions the system adapted itself rightfully and which conditions may lead to premature adaptations due to misinterpretations of the absence of coordination information.

Future work aims at the reimplementation of both the decentralized coordination framework for MAS (*DeCoMAS*) as well as the structural adaptation extension. It is envisioned to realize a more general coordination framework that does not only supports MAS but component-based applications in general. Therefore, the focus will shift from the coordination of agents to the coordination of universal software components. This should support a broader range of applications and both unify and simplify the engineering and development process of self-organizing application relying on decentralized coordination processes. This new Decentralized Coordination Framework (*DeCoF*) will be used to realize the self-organizing redistribution of bikes in a bike-sharing system as described in [35]. Based on the structural adaptations it shall be shown how the adaptation of different coordination processes optimizes the self-organizing redistribution of the bikes depending on the time of day and rush-hour situations.

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] T. Preisler and W. Renz, "Structural adaptations for self-organizing multi-agent systems," in Seventh International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE), Nice, France, 2015, pp. 1–8.

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

[3] 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.

[4] 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.

[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] A. Ismail and V. Cardellini, "Decentralized planning for self-adaptation in multi-cloud environment," in Advances in Service-Oriented and Cloud Computing, ser. Communications in Computer and Information Science, G. Ortiz and C. Tran, Eds. Springer International Publishing, 2015, vol. 508, pp. 76–90. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-14886-1_9

[24] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998.

[25] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, 1992, pp. 279–292.

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

[27] ——, "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.

[28] 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.

[29] 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.

[30] 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.

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

[32] L. Braubach and A. Pokahr, "Jadex active components framework - BDI agents for disaster rescue coordination," in Software Agents, Agent Systems and Their Applications, ser. NATO Science for Peace and Security Series - D: Information and Communication Security, M. Essaaidi, M. Ganzha, and M. Paprzycki, Eds. IOS Press, 2012, vol. 32, pp. 57–84. [Online]. Available: http://dx.doi.org/10.3233/978-1-60750-818-2-57

[33] A. Vilenica and W. Lamersdorf, "Simulation management for agent-based distributed systems," in Enterprise Information Systems, J. Filipe and J. Cordeiro, Eds. Springer-Verlag, 3 2011, pp. 477–492.

[34] ——, "Benchmarking and evaluation support for self-adaptive distributed systems," in The Sixth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS 2012), L. Barolli, F. Xhafa, S. Vitabile, and M. Uehara, Eds. IEEE Explore, 7 2012, pp. 20–27.

[35] T. Preisler, T. Dethlefs, and W. Renz, "Data-adaptive simulation: Co-operativeness of users in bike-sharing systems," in Proceedings of the Hamburg International Conference of Logistics, W. Kersten, T. Blecker, and C. M. Ringle, Eds., vol. 20. epubli GmbH, 2015.