Design and Simulation of a Wave-like Self-Organization Strategy for Resource-Flow Systems

Jan Sudeikat,
Wolfgang Renz, Thomas Preisler and Peter Salchow
Multimedia Systems Laboratory (MMLab),
Faculty of Engineering and Computer Science,
Hamburg University of Applied Sciences,
Berliner Tor 7, 20099 Hamburg, Germany
Email: {jan.sudeikat, wolfgang.renz,
thomas.preisler, peter.salchow}@haw-hamburg.de

Jan-Philipp Steghöfer, Hella Seebach
and Wolfgang Reif
Institute for Software and Systems Engineering,
Augsburg University,
Universitätsstrasse 6a, 86135 Augsburg, Germany
Email: {steghoefer, seebach,
reif}@informatik.uni-augsburg.de

***Abstract*—In resource-flow systems, e.g. production lines, agents are processing resources by applying capabilities to them in a given order. Such systems profit from self-organization as they become easier to manage and more robust against failures. This paper proposes a decentralized coordination process that restores a system's functionality after a failure by propagating information about the error through the system until a fitting agent is found that is able to perform the required function. The mechanism has been designed by combining a top-down design approach for self-organizing resource-flow system and a systemic modeling approach for the design of decentralized, distributed coordination mechanisms. The systematic conception of the inter-agent process is demonstrated. Evaluations of convergence as well as performance are performed by simulations.**

I. Introduction

A key driver in the development of autonomous and autonomic systems is the handling of complexity in large applications that consist of a great number of interacting entities. Traditional management and failure-handling approaches are no longer applicable as they do not scale well with the size of the systems and the communication required by a central management becomes prohibitive, even with modern high-speed networks. Therefore, engineers and computer scientists turn to self-organization as a means to deal with large complex systems and to keep up with the growth of such applications.

In this paper, we present a self-organizing process for the class of self-organizing resource-flow systems. This class can be applied to a great variety of domains such as production automation and logistics and systems in it can be modeled with the Organic Design Pattern (ODP) [1]. The decentralized process proposed here is analyzed and modeled with the tools provided by the SodekoVS project [2]. Changes in the configurations of agents propagate through the system like a wave until the system in its entirety has restored a stable state. During reconfiguration, parts of the system that are not affected by the process or have already been reconfigured are still able to resume their normal work. Evaluations show the quick convergence to stable states and the reconfigurations only affect system partitions.

This paper also shows how to pragmatically combine a top-down approach for the design of agent-based systems with a bottom-up approach for the design of inter-agent coordination. While the exact interplay of the two concepts is not fully elaborated here, it already becomes clear that both approaches are not necessarily orthogonal but that it is beneficial to combine both views.

This paper is structured as follows: in the following section, the ODP, as a conceptual model for self-organizing resource-flow systems, is discussed and a prominent application scenario, i.e. production automation, is introduced. In Section III, a programming model for self-organization is introduced. Subsequently, the intended coordination dynamics of self-organizing resource-flow systems are presented (see Section IV) and the realization of a decentralized role allocation strategy is discussed and evaluated (Section V). Finally, we conclude and give prospects for future work.

II. Design of Self-Organizing Resource-Flow Systems

In production automation systems, resources are transported between machines to subject these work peaces to a specific sequence of work steps. The sequence of machines is typically static. The machines that process the resources are highly specialized and only have one particular capability, i.e. an individual operation, they can apply to the resources. The transport of resources is fixed as well, e.g. by a static layout of conveyor belts. This rigid structure simplifies the management but has far-reaching implications, since reconfigurations are obstructed. The complete system has to be halted when internal errors make a single system component inoperable. Adjustments of the production process have to be carried out by stopping the system and retooling machines.

A visionary alternative are flexible, agent-based production lines that enable failure tolerance. Machines can autonomously reconfigure and transports of resources are carried out by *autonomous guided vehicles* (AGVs). Such a scenario is depicted in Fig. 1 where robots process a car body which is transported between the processing stations by autonomous carts.

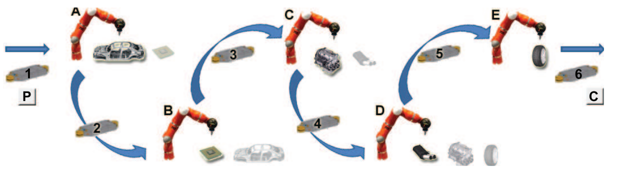


Fig. 1. Robots with different capabilities (icons to the right of the robot) process a car by applying one of their capabilities each (highlighted icon).

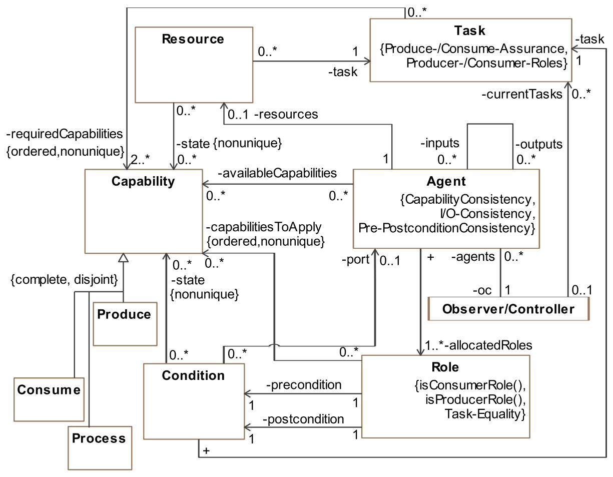


Fig. 2. The elements of the ODP for Resource-Flow Systems.

There are other domains where similar resource-flows occur, e.g., logistic scenarios or web service orchestration, we call this class of systems *Self-Organizing Resource-Flow Systems*. Their basic structure can be described with the ODP [3] which defines the elements that constitute the system and their relationship as shown in Fig. 2. *Task* define the required processing of *resources*. Processing steps are carried out by *agents*. The states of resources are modified by applying *capabilities*. Agents have a set of capability available and exchange resources, based on the shop layout (*inputs* and *outputs*). Which capability an agent applies and with which agents it exchanges resources is determined by a *role*[1]. Roles have a precondition that describes where the resource is coming from, which state it has, and which task has to be performed on it. They also have a postcondition that describes to which agent the resource has to be given and which state and task it has after the agent has processed it. Most importantly, the role defines the *capabilitiesToApply*, i.e., what an agent is supposed to do with the resource.

To fulfill the tasks for a resource (i.e., to apply the correct capabilities in the correct order), a resource-flow is established by the allocation of roles to agents that determines how the resource is moved through the system and processed on the way. This means the combination of the roles of the agents by their pre- and postconditions respectively, is a connected chain of agents along which resources in the system are forwarded and processed. There is usually one such chain or resource-flow for each task that has to be fulfilled in a system. Each agent, however, can participate in more than one resource-flow and thus be involved in several tasks at the same time.

The interactions between the agents to handle resources are also defined on the abstract system class level and can be inherited by applications based on the ODP. They describe, amongst other things, the handover of resources and the detection of agent failures with a heartbeat mechanism. This way, both a formal analysis of the system class [3], generalized mechanisms to deal with problems in the system class such as deadlocks [4] as well as a generic runtime environment [5] become feasible.

The ODP also contains an *Observer/Controller* (O/C). This element of the system structure is the abstract extension point for the self-organization or reconfiguration mechanism. Correct system behavior is defined by invariants that have to hold during the entire runtime of the system. Whenever the invariants are violated, the system has to be reconfigured to fulfill the invariants again (*Restore Invariant Approach* [3]). The individual agents are able to monitor local invariants and thus implement the observation part of the O/C. The controller part of the O/C is then responsible to calculate a new allocation of roles that restore the resource-flow and ensure that each agent has a role that fits its capabilities and its input/output relationship with other agents. How this calculation is done, however, is not specified at this point.

[1]Please note that some of the terminology used in ODP has a slightly different semantics than the same terms in agent-oriented software engineering due to the historic roots of ODP.

III. Systemic Programming of Self-Organization

In the research project "Selbstorganisation durch Dezentrale Koordination in Verteilten Systemen"[2] (SodekoVS) [2], a programming technique is developed that allows to equip software systems with self-organizing features. The self-organizing inter-agent process is described by discrete design elements [6]. This enforces a conceptual separation of the agent functioning and the coordination, i.e. the correlation of agent activities.

First, a modeling level for the description of inter-agent self-organization is provided. This modeling level supplements agent-oriented software engineering practices with an orthogonal description level that concerns the dynamic properties of agent-based software systems [6]. The driving force of self-organizing dynamics are distributed feedback loops among system elements [7]. These result from the mutual influences among system elements and control how fluctuations in the system context are disseminated and collectively responded to. The *systemic* modeling level addresses the description of these networks of influences and it has been found that the visualization of the mutual interdependencies of system elements is useful for the anticipation of the dynamics that software systems are able to exhibit [8], [6]. Using a graph-based modeling approach, *System Dynamics* [9] modeling concepts are specialized for describing Multi-agent systems (MAS).

[2]Self-Organisation by Decentralized Coordination in Distributed Systems

These models are given as an *Agent Causal Behavior Graph* (ACBG) [10]. The nodes in this graph-based modeling level represent system variables that characterize the macroscopic state of a MAS. These describe the number of agents that show a specific behavior, e.g. play a role. In addition, the current value of an interaction rate can be denoted with a specific node type. The links among these variables denote mutual influences and interdependencies. In this respect, *influences* denote additive or subtractive contributions to node values, e.g. when the activity of an agent increases or decreases the stock of a warehouse. *Interdependencies* describe causal relations where the activities of agents are mutually linked, e.g. the number of hypothetical service requesters in a system is expected to be positively linked to the number of activations of service providers. When the number of requesters increases, the number of activations increases as well and vice versa.

Secondly, a programming model that allows the enactment of ACBG-based prescriptions of self-organization processes [11], [10] facilitates application development. The key element is a distributed architecture for the enactment of decentralized inter-agent processes (cf. Fig. 3) [11]. This architecture serves as a reference model for the integration of ACBG-based processes in MAS. It provides a conceptual framework for fitting in different self-organization mechanisms and follows a layered structure. The topmost layer (*Application Layer*) contains the realization of an agent-based application. The contained agents are understood as self-contained providers of functionalities (*Application Functionality*). The contained agents individually control their activities and an underlying *Coordination Layer* enables the purposeful affecting of agents to concert the localized activities and establish collective behaviors.

The Coordination Layer describes an *event-based distributed system* [12], which allows to realize mutual influences among system elements. These influences correspond to relations in ACBG-based models of inter-agent processes, thus the layer is a means to enact the described processes in MAS. The establishment of inter-agent influences, particularly for the construction of self-organizing systems, is based on two types of mechanisms [13], i.e. techniques for the information exchange among agents (e.g. reviewed in [14]) and mechanisms for the (adaptive) adjustments of agents (among others classified in [15]), due to the perceived information (see Section VI). The Coordination Layer contains two types of functional elements for the encapsulation of these aspects. *Coordination Media* are conceptual containers of so-related interaction infrastructures. Specific interaction modes, e.g. the mediation by an environment [16] or Linda-like tuple spaces, are encapsulated and reused by a generic interface [11]. *Coordination Endpoints* interact on behalf of agents via these media and are able to influence the agent execution. These elements are used to encapsulate and automate the coordination-related activities. These activities concern the interactions vie Media, i.e. the invitation and participation of interactions, as well as the affectation of modifications in the agent models.

The ACBG-based modeling of dynamics of inter-agent coordination is exemplified in the Sections IV and V. A configuration language [10] allows to map ACBGs to agent-based software systems. These mappings describe the realization of influences among agents, i.e. the coordination-related logic that controls the initiation, participation, and reaction to interactions as well as the media that mediate interactions. The detailing of these models, as a systematic programming effort, is not discussed in this article but details on the configuration of process enactments can be found in [17].

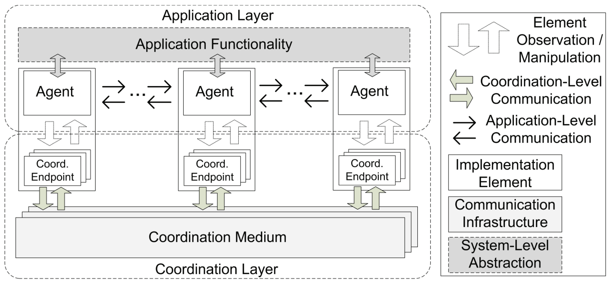


Fig. 3. The SodekoVS-Architecture for the embedding of decentralized coordination in MAS [17].

IV. Systemic Model of Adaptation Dynamics

In [18], the systematic integration of decentralized coordination strategies in MAS has been discussed. The conception of the appropriate coordination is approached by modeling the problematic, unintended behavior of applications. Based on the identification of the *Problematic Dynamic*, a corresponding *Solution Dynamic* is derived that supplements the application behavior with additional interdependencies and inter-element feedbacks to correct the system behavior and alleviate unintended effects.

The Problem Dynamic of an ODP-based resource-flow systems is illustrated in Fig. 4 (right). Initially, agents are *running* and one or several roles are allocated to them which are executed in order to process resources. Random errors make it impossible for the agent to apply one or more of its roles. The adoption of roles that can not be applied is controlled by a fluctuating rate (*RF interrupt*) that is positively influenced by the availability of running, thus breakable, agents and the changing number of error events (*Error*). This rate describes the *resource-flows* (RF) that are interrupted, due to the breaking of agents. These failures within individual agents limit the number of running agents (negative link), thus the problematic system behavior is dominated by a negative feedback loop (α).

If not handled, this dynamic causes the number of agents that are not running to increase over time. The design of an appropriate Solution Dynamic concerns the derivation of agent behaviors that counteract this unintended effect. A very general structure is given on the left hand side of Fig. 4. Agents that have roles they can no longer apply are *Waiting for Reconfiguration*. The rate of interrupts positively influences the increase of this variable. The system is equipped with a reconfiguration mechanism, and for each of the waiting

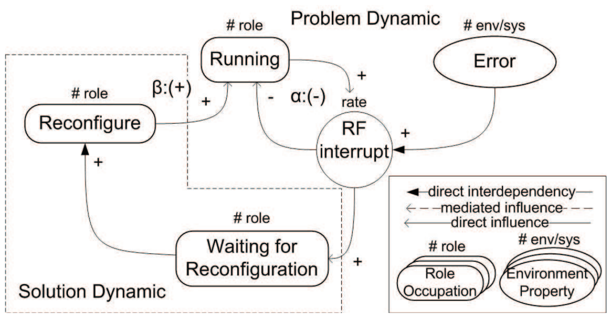


Fig. 4. The Problem and Solution Dynamic of the ODP.

agents a new configuration is determined. Thus the system shows a causal relation. In absence of waiting agents, no reconfigurations take place. Occurrences of waiting agents enforce subsequent reconfigurations (*Reconfigure*) to restore a set of executable roles. The reconfigurations thus increase the number of *Running* agents by complementing a counteracting feedback loop (β).

This Solution Dynamic deliberately omits the concrete mechanism with which new role allocations are determined. Also the locally applied techniques to the enactment of reconfigurations are abstracted. A method to express the problem of finding a fitting role allocation as a constraint-solving problem has been presented in [19] and solved with a centralized approach. Whenever an agent can no longer apply one of its roles or whenever an agent breaks, the resource-flow is interrupted. When the interruption is detected, the system reconfigures in order to restore the flow. During the course of the reconfiguration process, a new allocation of roles to agents is calculated and the roles are communicated to the agents which then apply them again. The next section describes an alternative reconfiguration mechanisms in which new role allocations are found in a decentralized fashion by propagating the demand for local reconfigurations through the system.

V. Wave-like Decentralized Reconfiguration

A completely decentralized reconfiguration approach is based on the idea that a wave of role re-allocation runs through the system in order to re-establish the resource-flow. Assuming that each agent is capable to exhibit a set of capabilities (see Section II), a correct resource flow can be (re-)established by the appropriate swapping of roles. Failing agents adopt actable roles and in return other agents help out by providing the unactable roles. The failing agent emits a wave of reallocations by sending requests for assistance along the resource flow. Each recipient has to decide locally if it is capable and will swap roles. Generally, a single swap of roles is not enough to reestablish the full sequence of activities and transitive changes of roles are required.

The operating principle is exemplified in Fig. 5 for a simple scenario that requires a transitive swap of roles. Three Robots are connected in series (1). For each agent the set of actable capabilities are listed and the topmost capability is used in the currently active role, e.g. *Robot 1* is currently configured to play a role that involves *Cap. 1* but may also apply *Cap. 3*. Two types of Cart agents represent the initial provision (*Producer*) and the final collection of the processed workpieces (*Consumer*). Due to an error *Robot1* can no longer apply *Cap. 1* and sends a request for assistance (2). This request is routed along the resource flow till it reaches an agent that is capable to execute the needed capability, here *Robot 2* which replies to the request (3). The reply is routed to the requesting Robot as well as the Carts that are connected to the swapping robots. Consequently, the Robots and Carts reconfigure their local roles. The robots update their roles and the resource flow is reestablished by adjusting the ports in the pre- and post-conditions of the roles of the connected Carts to ensure that workpieces reach the robots in the intended sequence. In the best case, the originating and the receiving agents can just switch their roles, thus restoring the resource-flow.

In Fig. 5, *Robot 2* is able to provide capability *Cap. 1* but *Robot 1* is not able to replace *Robot 2* as it is missing the currently utilized *Cap. 2* (3). Thus after the swap, *Robot 1* remains in a problematic state and requests assistance (4). This request is propagated again till it reaches a robot with the required capability (*Robot 3*) and the swap proceeds as above (5). Since *Robot 1* is able to replace the currently active capability of *Robot 3*, i.e. *Cap. 3*, the correct sequence of capabilities is finally reestablished (6). Here, the reconfiguration logic has been described for agents that only play on role at a time and the subsequent simulations concerns this simplified scenario. In principle, agents can be part of several resource-flows and in that case, the agents only reconfigure for roles that include the broken capability and keep processing resources of other tasks. Consequently, the informed Carts change only the ports of the affected roles accordingly.

A detailed ACBG of the outlined reconfiguration algorithm is illustrated in Fig. 6. This description refines the previously given Solution Dynamic (cf. Fig. 4) as it illustrates how the decentralized strategy relates to the dynamics of ODP. In addition, it indicates the system-level effects of the decentralized reconfiguration that are examined in Section V-A. When agents are *Waiting for Reconfiguration* due to error events, they show two behaviors. First, they are *Deficient* as one or more roles, which are required for the processing of resources, are inoperable. These roles are distinguished by the reason for deficiency. Agents can be rendered deficient by error events (*By Break*) or they deliberately decided to abandon a role in order to adopt another role on behalf of another agent (*By Change*). In the former case, the agents are rendered inefficient and in the latter case agents assist other agents. Secondly, these agents are considered to be *Non-Active*, i.e. they have the capacity to play another role. Agents can concurrently play several roles, therefore, the non-activity only denotes that the agent is underutilized, i.e. is capable to exhibit another role. This means that in agents that can play several roles, the Running and Non-Active roles do not exclude each other, but an agent can be associated to an active role (Running) and still have the capacities to play additional roles (Non-Active).

Agents are equipped with the ability to autonomously

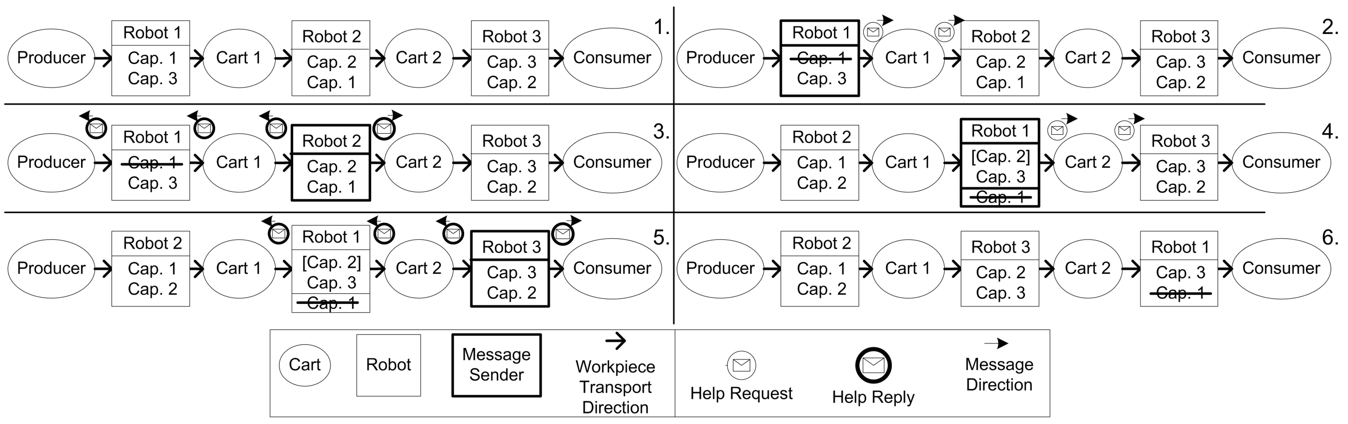


Fig. 5. Exemplification of the decentralized reconfiguration.

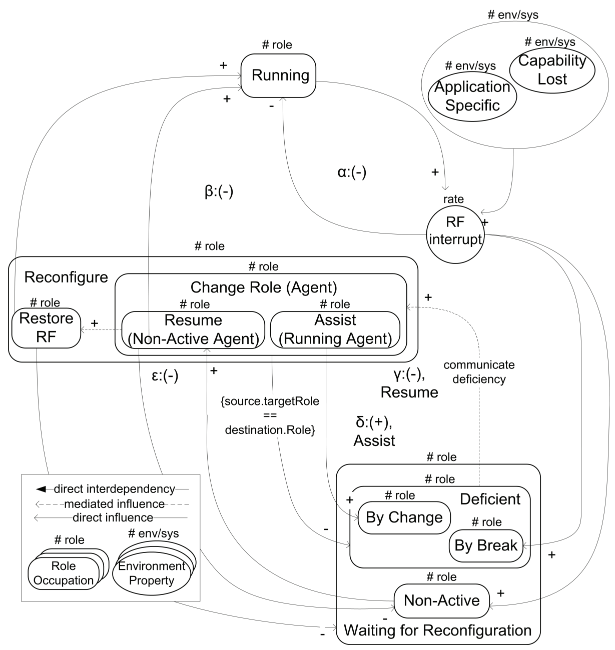


Fig. 6. ACBG for the Solution Dynamic of the wave-like, decentralized reconfiguration algorithm

change their allocation (*Change Role*). Deficient robots indicate their shortcoming to other agents (*communicate deficiency*) via a Coordination Medium (cf. Section III). The medium controls the sequential reception of the request along the flow of resources in the production line. Recipients decide locally whether to change their role-allocation or not, based on their individual abilities. The changing behavior is distinguished by the receiving agent that adjusts the local configuration. Non-Active agents *Resume* the executions since these become operational. Running agents that adjust their role *Assist* the requesting agent. These roles have different effects on the agent population. All changes remove deficiencies and the annotation *source.targetRole == destination.Role* indicates that only those deficiencies are removed (destination) that changing agents (source) commit to. Another commonality is that the adjustment of a role entails the restoring of the flow of resources among the agents (*Restore RF*). When an agent has adjusted its role, those agents that received resources from it or gave resources to it need to adapt as well. This activity is separated from the role change as the agents do not deliberately decide about these changes. These are reconfigurations within connected agents that are enforced as they are consequences of the deliberate changes. These reconfigurations may as well by transmitted via Coordination Media (see Section III).

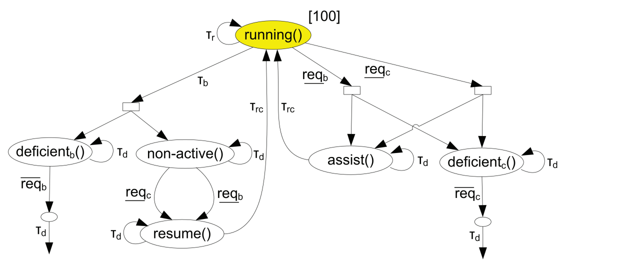
Assisting another agent introduces new deficiencies, as the assisting agent is giving up one role that needs to be played by another agent (*Deficient by Change*). Thus the net amount of Non-Active agents is unaffected. However, these changes may be necessary in settings where agents can not swap roles directly and transitive changes of roles are required (as exemplified in Fig. 5). When Non-Active agents *Resume* to adopt a role, the number of Non-Active agents is reduced. Still, this requires the availability of Non-Active agents.

The changing activities of agents control the overall negative feedback (β) that increases the number of operational agents and reduces the number of deficient agents. Three auxiliary feedbacks influence the exhibited system dynamics (δ,γ,ϵ). First, agents that assist others create a reinforcing feedback loop (δ), which originates from the fact that an *assisting* agent adds additional deficiencies to the system. If an agent *resumes* its activities, deficiencies and non-active agents are removed instead, thus instituting a balancing feedback (γ). The ability to resume is limited by the amount of Non-Active agents, leading to a third balancing feedback (ϵ).

A. Estimating the Effects of Adaptation Dynamics

The outlined Solution Dynamic (cf. Fig. 4) denotes an inter-agent process that can be implemented with the systemic programming model (see Section III). Prior to the detailed design and embedding of this process, the effected system behavior is anticipated. One approach to estimate the dynamics of self-organizing MAS is their simulation in stochastic process algebra models [20]. It is important to note that the resulting stochastic models abstract from the agent implementations and their internal reasoning. Instead, stochastic terms are used to describe the dynamics with which specific behaviors are adopted or left. The number of currently active process terms resembles the number of agents that show specific behavior, i.e. the variable-values of an ACBG-based process description.

Fig. 7 illustrates the simulation model used to anticipate the Solution Dynamic. The model is given in stochastic π-calculus [21] and simulated with the *Stochastic Pi Machine*[3] (SPIM). Details on the simulation language and graphical notation can be found in [22]. Agents are modeled as processes that communicate/interact via channels. The stochastics of interactions are given by annotating processes with delays (τ) and assigning interaction rates to channels [21], [22]. The notation $\overline{x}$ denotes the sending of data on the channel x and $\underline{x}$ denotes the reception of data via the channel x.



$$\begin{aligned}
\text{running}() &= \tau_r.\text{running}() + \\
&\quad \underline{req}_b.(\text{deficient}_c() \mid \text{assist}()) + \\
&\quad \underline{req}_c.(\text{deficient}_c() \mid \text{assist}()) + \\
&\quad \tau_b.(\text{deficient}_b() \mid \text{idle}()) \\
\text{resume}() &= \tau_{rc}.\text{running}() + \\
&\quad \tau_d.\text{resume}()
\end{aligned}$$

$$\begin{aligned}
\text{deficient}_b() &= \tau_d.\text{deficient}_b() + \\
&\quad \overline{req}_c.\tau_d \\
\text{non-active}() &= \tau_d.\text{idle}() + \\
&\quad \underline{req}_b.\text{resume}() + \\
&\quad \underline{req}_c.\text{resume}()
\end{aligned}$$

$$\begin{aligned}
\text{deficient}_c() &= \tau_d.\text{deficient}_c() + \\
&\quad \overline{req}_c.\tau_d \\
\text{assist}() &= \tau_{rc}.\text{running}() + \\
&\quad \tau_d.\text{assist}()
\end{aligned}$$

Fig. 7. Simulation Model of the Solution Dynamic in Stochastic π-Calculus.

In the simulation model, the number of agent behaviors that are exhibited are expressed by the number of active processes. Processes communicate via two channels. The channel req_b is used to communicate requests for a role-switch, due to an internal error, i.e. the internal breaking of the requester. When roles are requested to be switched in order to assist an agent, these requests are send via the channel req_c. Operative agents are denoted by the $running$ process. Internal errors occur with a fixed rate, as defined by the delay τ_b. Inoperative robots are represented the concurrent execution of the $deficient_b$ and *non-active* processes. Deficient processes end when a request for re-assignment of the affected role is processed by a recipient. The *non-active* processes become *resume* processes when they receive any request for re-allocation. The time delay τ_{rc} resembles the time needed to reconfigure. Afterwards, the agent is in operation, i.e. exhibits the $running$ process. When $running$ agents receive a request to switch roles, they convert to the concurrent execution of the $assist$ and $deficient_c$ processes. The assistance transforms back to the running process, delayed by the time to reconfigure the agent (τ_{rc}). The $deficient_c$ processes describe the search for another agent that is able to play the role that an assisting agent possessed before the assisting adjustment. Deficiency ends when another agent processes the request for a role change, communicated via the channel req_c. This simulation model abstracts from the agents that participate in the system. The time needed to *restore the resource flow* (cf. Fig. 6), i.e. the adjustments of the roles of the directly connected agents to reestablish the correct flow of resources among machines, is part of the time delay τ_{rc}. We assume that the rerouting of resource transportations is always possible.

Simulations indicate that, at a high enough level of redundancy, the system reliably recovers due to the decentralized switching of agent roles. This process describes a structure formation as the system maintains the operational system configuration. The fraction of recovering situations is predicted by this simulation to depend on the redundancy level in a similar way as is shown in the results section below.

B. Agent-based Realization

After the anticipation of the affected system behavior, this reconfiguration strategy has been integrated in a MAS by using systemic programming model (see Section III). The system implementation (*Application Layer*, see Figure 3) makes use of the freely available *Jadex*[4] agent framework. The Robots and Carts within production lines are represented by Jadex-agents and the exchanged workpieces are mimicked by objects that are exchanged via FIPA *Agent Communication Language* (ACL) messages. A realization of the *Coordination Layer* (see Fig. 3) for this agent platform is utilized [11].

For this application scenario a tailored Medium realization is utilized that routes request and reply messages along the resource flow. Conceptually though, agents are aligned in a circle thus all agents can be reached independent of the location of the incapacitated agent. Endpoints encapsulate the logic to coordinate the reconfiguration process and interact via the Medium. Endpoints observe the agent-operation and initiate the reconfiguration process by sending a help request if an agent becomes deficient. The help request is forwarded through the medium. Each endpoint along the message path decides whether to adopt the deficient agent's role or continues forwarding the help request. If the endpoint decides to adopt the role, a reply is sent. The reply is sent in both directions through the medium to inform all agents which are affected (robots and connected carts) by the reconfiguration process. Again, each endpoint receiving the reply decides to change the agent configuration. The reply is sent backward through the medium until all affected agents are informed. If an endpoint receives multiple coordination messages these messages are queued and processed in the order of their arrival.

C. Implementation Test Results

The example system illustrated in Fig. 5 has been implemented and tested for measuring the handling of breaking capabilities. When the robot is rendered incapacitated by such an event, the associated endpoint notices this and initiates

[3]http://research.microsoft.com/en-us/projects/spim/

[4]http://jadex-agents.informatik.uni-hamburg.de/

an interaction via the Coordination Medium that triggers the swapping of roles. The first swap involves the incapacitated agent that is *Deficient by Break*. In this system configuration, a second swap is required that resolves a transient *Deficient by Change* agent behavior.

In addition, we examine the relation of the redundancy within agents with the effectiveness of the reconfigurations. The effectiveness of the reconfiguration procedure is influenced by the number of alternative capabilities that are available to the individual agents. This level of redundancy is measured by the ratio of the number of *individual* capabilities (C_i), to the *absolute* number of capabilities (C) that are required for the processing of a workpiece ($\frac{C_i}{C}$). In the following, we assume a homogeneous setting, where the robots are equipped with an equal number of redundant capabilities. The composition of these capabilities is normally distributed. In Fig. 8, measuring results for a simple scenario with 10 different capabilities and agents are shown. Each single run processes of a fixed number of workpieces while, at two fixed instances of time, a randomly selected agent is incapacitated. A first estimate of the effectiveness of the reconfigurations is the number of exchanged messages (see Fig. 8). The number of messages increases quickly when the number of redundancies decreases. This measurement can be analytically fitted with $(c_1 * (1-x))^2 + c_2$.[5]. A complementary measurement is the number of hops that requests for assistance travel before a swapping agents is found. This describes the logical distance between the swapping agents. The results for this measurement have shown the same characteristics like the message count and can be fitted with the same function (but different constants). Thus the decentralized reconfiguration strategy is particularly suited for production lines where the capability types are often available.

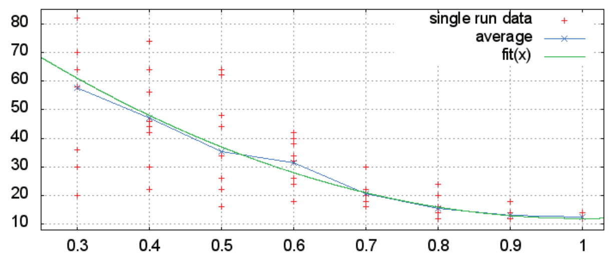


Fig. 8. Measurement results. The averaged number of messages are plotted over the redundancy of capabilities.

VI. Related Work

Ant Colony Optimization has been used for decentralised control in production systems. [23] uses the mechanism to control autonomous vehicles, similar to our carts. The distribution of jobs to production machines has not been a concern there. The more complex problem of scheduling jobs to run on certain machines has, e.g., been tackled in [24] and [25]. However, these papers from operations research focus on optimization of a shop floor and do not take into account robustness and reconfiguration that happens at run-time. Also, only partial problems are investigated, either focusing on the routing of carts or the scheduling of production machines.

The work presented here concerns the run-time reconfiguration by self-organization. The maintained structure is a correct sequence of agents that are perturbed by individual failures that incapacitate agents. Prominent alternatives to the process presented here are centralized/distributed constraint solving techniques and market-based mechanisms. The correct configuration is described with constraints and an approach to use centralized constraint solving to restore functionality after a failure has already been published [19]. Consequently, distributed constraint solving techniques [26] can be used as well, e.g. as studied in [27]. An example for market-based mechanisms is given in [28]. A manufacturing line is provided with a flexible transport mechanism and work pieces and machine agents negotiate for the execution of working steps. In a way, the algorithm proposed in this paper is an *optimistic* and *minimalistic* version of a distributed constraint solver. By exchanging roles, the agents collectively restore the invariants of the system. The strategy is minimalistic since the number of required messages and the amount of shared information is reduced. It is optimistic because the assisting role changed are carried out before the complete solution is calculated. Its main advantage over traditional distributed constraint solvers is the minimal amount of calculation that is involved at the agents. They can therefore be very small with only minimal CPU power and RAM, making them cheap and easily replaceable.

Here, these design alternatives are qualitatively characterized by a subset of criteria from [29]. Their quantitative comparison is left for future work. A first aspect is the necessary *communication*. The presented process minimizes the message content and only the information that immediately necessary to resolve a local failure is communicated. The number of messages ranges from a single role swap (best case) to the successive swapping of roles by all agents (worst case). The dependence of this measure on structural properties is shown in Figure 8. The communications are only carried out when failures are present. Alternatives involve that coordination-related messages have to be exchanged during the normal system operation, e.g. as workpieces constantly negotiate their further processing [28]. Also the amount of *computations* and the considered/exchanged *knowledge* about the system state is minimized. The participation in the process involves only the local consideration whether an agent is capable to play a required role. Now further information about the global system state is processed. Centralized constraint solvers require the full knowledge about the system and decentralized solvers require the information from neighboring agents. In addition, the *adaptations* are carried out concurrently to the system operation. Unaffected partitions of the production line continue to work. Finally, the accuracy of the quality of the found solution, i.e. stable configuration, varies. The process follows the heuristic that role swaps of nearby agents are favored over

[5] c_1, c_2 are application dependent constants. In this case, these are 10 and 12 respectively.

the swaps of (logically) distant agents. The explicitly treatment of the underlying constraint problem allows in principle to prepare the optimization of the found solutions.

VII. Conclusions

In this paper, we have described a decentralized reconfiguration process to restore valid system configurations in self-organizing resource-flow systems. The reconfiguration algorithm works by exchanging roles with neighboring agents and by propagating change requests in a wave-like manner until all of them could be satisfied. The mechanism has been developed by combining a top-down process for the description of resource-flow systems and a bottom-up process for the design of agent coordination. Its performance has been demonstrated with a number of simulations.

The most interesting feature of the decentralized process proposed here is that reconfigurations are organized locally in the production line, i.e. the rest of the system is not impaired by a failure. Thus, parts of the system that are not involved into a local reconfiguration can continue to run normally. The way the reconfiguration propagates also ensures that only a small amount of agents is in a state of non-processing resources at an instance of time. This feature will be prominent also when using the wave-like algorithm in non-linear production situations, which is a straight-forward generalization instance. Local heuristics taking into account exchange–success rates could be predefined or evolved using learning algorithms. Future work includes a more detailed study on the combination of bottom-up design of coordination methods as proposed in SodekoVS and of top-down design methodologies as promoted with the ODP. This will also include a comparison of their respective advantages and problems that occur when both worlds are combined.

Acknowledgment

This research is partly sponsored by the German research foundation (DFG) in the project SodekoVS and in the DFG special priority program "Organic Computing" (SPP 1183) in the project SAVE ORCA.

References

[1] H. Seebach, F. Ortmeier, and W. Reif, "Design and Construction of Organic Computing Systems," *IEEE Congress on Evolutionary Computation, 2007*, pp. 4215–4221, Sept. 2007.

[2] J. Sudeikat, L. Braubach, A. Pokahr, W. Renz, and W. Lamersdorf, "Systematically engineering selforganizing systems: The sodekovs approach," *Electronic Communications of the EASST*, vol. 17, 2009.

[3] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif, "A specification and construction paradigm for Organic Computing systems," in *Proceedings of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2008, pp. 233–242.

[4] J.-P. Steghöfer, P. Mandrekar, F. Nafz, H. Seebach, and W. Reif, "On Deadlocks and Fairness in Self-organizing Resource-Flow Systems," in *Proceedings of the 30th International Conference on Architecture of Computing Systems (ARCS), LNCS 5974*. Springer, 2010, pp. 97–100.

[5] F. Nafz, F. Ortmeier, H. Seebach, J.-P. Steghöfer, and W. Reif, "A generic software framework for role-based Organic Computing systems," in *SEAMS 2009: ICSE 2009 Workshop Software Engineering for Adaptive and Self-Managing Systems*, 2009.

[6] J. Sudeikat and W. Renz, "Qualitative modeling of mas dynamics - using systemic modeling to examine the intended and unintended consequences of agent coaction," in *Agent-Oriented Software Engineering X*. Springer, 2009, to be published.

[7] Y. Brun, G. di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, *Software Engineering for Self-Adaptive Systems*. Springer-Verlag, 2009, ch. Engineering Self-Adaptive Systems through Feedback Loops, pp. 48–70.

[8] W. Renz and J. Sudeikat, "Modeling feedback within mas: A systemic approach to organizational dynamics," in *Organised Adaptation in Multi-Agent Systems*, 2008, pp. 72–89.

[9] J. D. Sterman, *Business Dynamics – Systems Thinking and Modeling for a Complex World*. McGraw-Hill, 2000.

[10] J. Sudeikat and W. Renz, "MASDynamics: Toward systemic modeling of decentralized agent coordination," in *Kommunikation in Verteilten Systemen*, ser. Informatik aktuell, 2009, pp. 79–90.

[11] ——, "Decomas: An architecture for supplementing mas with systemic models of decentralized agent coordination," in *Proc. of the 2009 IEEE/WIC/ACM Int. Conf. on Intel. Agent Tech.*, 2009, pp. 104–107.

[12] G. Mühl, L. Fiege, and P. Pietzuch, *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., 2006.

[13] J. Sudeikat and W. Renz, *Applications of Complex Adaptive Systems*. IGI Global, 2008, ch. Building Complex Adaptive Systems: On Engineering Self–Organizing Multi–Agent Systems, pp. 229–256.

[14] T. DeWolf and T. Holvoet, "Decentralised coordination mechanisms as design patterns for self-organising emergent systems," in *Engineering Self-Organising Systems*, vol. 4335/2007, 2007, pp. 28–49.

[15] G. D. M. Serugendo, M. P. Gleizes, and A. Karageorgos, "Self-organisation and emergence in mas: An overview," in *Informatica*, vol. 30, 2006, pp. 45–54.

[16] H. V. D. Parunak and S. Brueckner, "Engineering swarming systems," in *Methodologies and Software Engineering for Agent Systems*. Kluwer, 2004, pp. 341–376.

[17] J. Sudeikat and W. Renz, "Programming adaptivity by complementing agent function with agent coordination: A systemic programming model and development methodology integration," *Communications of SIWN*, vol. 7, pp. 91–102, may 2009, iSSN 1757-4439.

[18] ——, "On the modeling, refinement and integration of decentralized agent coordination – a case study on dissemination processes in networks," in *Self-Organizing Architectures*, 2010, pp. 251–274.

[19] F. Nafz, F. Ortmeier, H. Seebach, J.-P. Steghöfer, and W. Reif, "A universal self-organization mechanism for role-based Organic Computing systems," in *Proceedings of the Sixth International Conference on Autonomic and Trusted Computing (ATC-09)*, 2009.

[20] M. Casadei, L. Gardelli, and M. Viroli, "Simulating Emergent Properties of Coordination in Maude: the Collective Sorting Case," in *5th Int. Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA)*, 2006.

[21] C. Priami, "Stochastic π–calculus," *Computer Journal*, vol. 6, pp. 578–589, 1995.

[22] A. Phillips, *Symbolic Systems Biology: Theory and Methods*. Jones and Bartlett Publishers, 2010, ch. A Visual Process Calculus for Biology.

[23] V. A. Cicirello and S. F. Smith, "Ant colony control for autonomous decentralized shop floor routing," in *International Symposium on Autonomous Decentralized Systems*, 2001.

[24] N. Liouane, I. Saad, S. Hammadi, and P. Borne, "Ant systems & local search optimization for flexible job shop scheduling production," *Int. Journal of Comp., Comm. & Control*, vol. 2, no. 2, pp. 174–184, 2007.

[25] C. Gagné, M. Gravel, and W. L. Price, "Solving real car sequencing problems with ant colony optimization," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1427 – 1448, 2006.

[26] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 5, pp. 673–685, 1998.

[27] G. Clair, E. Kaddoum, M.-P. Gleizes, and G. Picard, "Self-regulation in self-organising multi-agent systems for adaptive and intelligent manufacturing control," in *SASO '08: Proc. of the 2008 Sec. IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems*, 2008, pp. 107–116.

[28] N. R. Jennings and S. Bussmann, "Agent-based control systems," *IEEE Control Systems*, vol. 23, no. 3, pp. 61–74, 2003.

[29] E. Kaddoum, M.-P. Gleizes, J.-P. Georgé, and G. Picard, "Characterizing and evaluating problem solving self-* systems," in *COMPUTATION-WORLD '09: Proc. of the 2009 Computation World*, 2009, pp. 137–145.