Adapting Environment-Mediated Self-Organizing Emergent Systems by Exception Rules

Holger Kasinger
University of Augsburg
Institute for Software and Systems Engineering
86135 Augsburg, Germany
kasinger@ds-lab.org

Bernhard Bauer
University of Augsburg
Institute for Software and Systems Engineering
86135 Augsburg, Germany
bauer@ds-lab.org

Jörg Denzinger
University of Calgary
Department of Computer Science
Calgary, Canada T2N 1N4
denzinge@cpsc.ucalgary.ca

Tom Holvoet
KU Leuven
Department of Computer Science
3001 Heverlee, Belgium
tom.holvoet@cs.kuleuven.be

ABSTRACT

Due to the absence of global knowledge, elements in a self-organizing emergent system tend to make suboptimal local decisions that result in globally inefficient solutions. However, improving the solutions of such systems, which work in a bottom-up style, by the principles of self-adaptive systems, which work in a top-down style, is not a straight forward process. In this paper, we present challenges and constraints that have to be respected during this process and describe early work on an approach, how to autonomously adapt the local behavior of self-organizing elements by so-called exception rules in order to improve the performance of the global solution. In particular, we present a set of exception rules that can be employed in different situations for the improvement of environment-mediated, self-organizing emergent solutions to pickup and delivery problems.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; D.2.11 [**Software Architectures**]: Patterns; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Performance

Keywords

Self-adaptation, self-organization, emergence, efficiency, rules

SOAR'10, June 7, 2010, Washington, DC, USA.

1. INTRODUCTION

Self-organizing emergent systems [6] consist of many locally interacting elements, which can either be autonomous software elements, such as agents [19], or autonomous real-world elements, such as mobile devices, robots, or modern cars [13]. In these systems, the elements (inter)act only locally and as a result dynamically acquire or maintain a global structure or behavior in a bottom-up style without external control. Usually, the elements are kept relatively simple, e. g. due to limited resources, have only partial or even no global system knowledge, and make their decisions solely based on local information available from their neighbors in the communication topology as well as their perception of the environment. These systems thus are known for their high flexibility, scalability, robustness, and adaptivity.

However, these systems are not known for their optimality. In particular when solving dynamic problems, the system elements tend to make 'suboptimal' local decisions that result in global inefficiencies. Please note that in order to make 'optimal' local decisions on its own, a system element would have to be in possession of an abundance of relevant information, including information about the environment topology (e. g. networks, machines, customers, ...), the current and future state of the environment, including all problem-relevant tasks, as well as the current and future intended behavior of other elements. This would not only force an element to quickly gather real-time information from a large number of possibly unknown entities, but also to be able to "look into the future", such that a dynamically appearing task can be assigned to the best element (with respect to the global optimality), while other tasks are already executed. Apparently, this would be a complex endeavor.

Thus, in order to assess the 'optimality' of their global solutions on their own and to change their structure or behavior when the assessment indicates that a better performance is possible, the top-down principles of self-adaptive systems [5] have to be added to self-organizing emergent systems. However, combining the principles of self-adaptive and self-organizing emergent systems in an appropriate system architecture requires to take into account several princi-

pal challenges and constraints, such as considering the openness and autonomy of an underlying self-organizing system, taking into account its low observability and poor controllability, and preserving its basic self-organizing and emergent behavior (see Subsection 2.1 on these challenges). In consideration of these challenges, in [20] we have presented the general model of the Efficiency Improvement Advisor (EIA). The EIA is able to autonomously adapt the local behavior of self-organizing agents in a self-adaptive manner, in order to improve the efficiency of the global solution. Subsection 2.2 briefly reviews the functionality of the EIA approach.

Whereas the exemplary instantiation of the EIA approach presented in [20] was able to successfully improve one particular inefficiency in a self-organizing emergent multi-agent system (MAS) solution – grounded on infochemical-based coordination (IBC) [10] – to Pickup and Delivery Problems (PDPs) [17], the purpose of this paper by contrast is to use the EIA approach for a broader set of inefficiencies as well as a broader class of systems. Thus, in this paper we present on the one hand a number of situations, in which agents in environment-mediated MAS solutions [22] in general tend to make 'suboptimal' local decisions that result in global inefficiencies (see Section 3). On the other hand, we present a number of so-called exception rules, which an EIA can employ to provide advice to misbehaving agents in these situations and which help them to make more 'optimal' local decisions (see Section 4). Section 5 explains the functionality of two of these exception rules in more detail, using a case study from the domain of PDPs again, whereas Section 6 presents conclusions and a number of open questions to be answered by future work.

2. IMPROVING THE EFFICIENCY OF SELF-ORGANIZING SYSTEMS

Adapting the local behavior of agents in order to improve the efficiency of the global solution requires to integrate a higher-level agent on top of the MAS that is equipped with additional functionality compared to the basic agents. However, in case of self-organizing emergent MASs the appropriate engineering of such hierarchical system architectures bears several challenges and constraints.

2.1 Challenges and Constraints

(1) Consider the openness and autonomy of the underlying system: A higher level agent has to adapt the behavior or structure of a basically open MAS autonomously depending on current, past, and future situations, which were potentially unexpected and unforeseeable at design time. This process must not limit the autonomy of the underlying system, i. e. all problem solving decisions must still be made locally by the agents on the lower level themselves.

(2) Take into account the low observability and poor controllability: A higher level agent will neither be able to observe all (inter)actions of lower level agents, if ever, nor be able to gather all relevant information at the time of occurrence. A higher level agent furthermore will neither be able to optimally adapt or influence the local behavior, intention, or upcoming action of any lower level agent yielding immediate effects, nor can guarantee that communication is flawless and that every agent 'hears about' its adaptation.

(3) Preserve the basic self-organizing and emergent behavior: A higher level agent may neither limit the basic system's self-organizing and emergent problem-solving behavior, nor limit its scalability, robustness, flexibility, or adaptivity. This implicates that the higher level agent may not act as a central controller and thus as a bottleneck and single point of failure. Consequently, if this agent crashes, the agents on the lower level still have to function properly.

2.2 Efficiency Improvement Advisor

Although there exist a couple of approaches in various fields that – similarly to the EIA approach – aim to adapt an underlying MAS by a higher level agent (see e. g. [4, 8, 11, 15, 18]), the higher level agent in these approaches very often either takes over central control in predefined situations, assumes to be able to observe and control the underlying MAS at every point in time, does not preserve the basic beneficial properties, or is unable to "look into the future". In order to review the EIA approach, first the concept of an agent as understood in this paper is defined and some assumptions regarding the problems to solve are made. For a more detailed description of the EIA approach we refer to [20].

2.2.1 *Definitions and Assumptions*

A very generic definition of an agent $\mathcal{A}g$ is as a 4-tuple $\mathcal{A}g = (Sit, Act, Dat, f_{Ag})$, where Sit is the set of situations the agent can face (i.e. its possible view of the environment), Act is the set of actions $\mathcal{A}g$ can perform, Dat the set of possible values of the agent's internal data areas and $f_{Ag} : Sit \times Dat \rightarrow Act$ the agent's decision function, describing how $\mathcal{A}g$ selects an action based on its current situation and the current value of its internal data areas (i. e. its perceptions of the world and its current knowledge status). This assumes that there is an action for every combination of activities the agent can do. A MAS is then a group of agents $A = \{\mathcal{A}g_1, \ldots, \mathcal{A}g_n\}$ that share an environment $\mathcal{E}nv$.

The general structure of problems that have to be solved by a set of agents A we focus on consists of tasks out of a set T that are announced to A at some times within a given time interval $Time$ to form a *run instance* for the system A. Usually, there will be a sequence of run instances that A has to solve. For instance, a run instance could be all the tasks A has to solve at a particular day, whereas a sequence of run instances are the tasks to solve over several days.

Users of a self-organizing emergent system A usually associate with a solution sol also a quality of the solution $qual(sol)$, which naturally depends on the application domain the agents in A have been designed for. Apparently, A is expected to produce a solution of optimal quality, which under some circumstances is easy, but under many circumstances is not, e. g. for NP-complete problems, in particular if a task ta can arrive at any point in time within $Time$. Since the agents do not know at the beginning of $Time$ what all tasks will be, it is in most cases impossible for A to solve a whole (dynamically developing) run instance optimally.

Furthermore, we assume the following conditions to be fulfilled: (1) Each agent $\mathcal{A}g$ is able to transmit a history of its local behavior to the advisor at least once during or after a run instance. (2) Each agent's decision function f_{Ag} can be extended to deal with exception rules (stored in its internal data area). (3) A sequence of run instances must have a (sub)set of similar tasks in (nearly) each instance of the sequence. While the first two conditions usually are achieved easily, the third condition seems very restrictive at a first glance. But in everyday life, there are many problems that

fulfill this condition, e. g. transportation companies usually have daily recurring tasks together with one-of-a-kind tasks.

2.2.2 Architectural Overview

The EIA is able to act autonomously, i. e. it improves the efficiency of the basic MAS without any user interaction based on a closed feedback control loop, but in contrast to existing approaches respects all of the challenges and constraints mentioned in Subsection 2.1. Basically, the EIA therefore performs six distinct functions connected by one data model (see Figure 1):

1. ***Receive* local agent histories:** The EIA collects the local histories of the lower level agents, i. e. mainly the situations they have perceived, the actions they have performed, and information on the environment, at least once during a run instance or at its end. This is essential, as due to the low observability of the system the EIA usually is neither aware of the tasks that have to be fulfilled nor the actions executed by the agents of the basic MAS. The EIA stores the collected local history of each agent in its data model. The interaction only occurs when communication is possible and does not interfere with the fulfillment of the agents' tasks.

2. ***Transform* local agent histories into global history:** Based on the received local agent histories the EIA then creates the global system history (and the system environment as far as possible). This provides the EIA with a (nearly) global view on the past sequence of run instances the agents had to solve.

3. ***Extract* recurring tasks from global history:** Based on the global system history the EIA then identifies recurring tasks patterns. Because in general tasks may slightly change between two or more run instances, the EIA not only identifies tasks that are identical in all run instances but also tasks for which similar tasks exist in all or at least most of the run instances. This set of recurring tasks apparently constitutes a core problem that appears repeatedly in the basic MAS.

4. ***Optimize* solution of recurring tasks:** The EIA then calculates a (nearly) optimal solution for the set of recurring tasks by means of exact optimization algorithms, heuristics, or metaheuristics. An optimization of the solution of all tasks that occur in the run instances is not necessary, because usually not all of these tasks will occur in future run instances again. By contrast, it can be supposed that at least the recurring tasks (or similar ones) will still occur in future.

5. ***Derive* rules from the optimal solution:** From the calculated optimal solution for the set of recurring tasks the EIA then derives advices in the form of exception rules, which can be of different types (see also later Section 4), for those agents that do not behave optimally in regard of the optimal solution. Therefore, the EIA first identifies and extracts the solution the emergent system has generated for this set of recurring tasks, determines differences between the emergent solution and the optimal solution, and, if applicable, creates exception rules for the misbehaving agents. Usually, exception rules are individual for each agent, while not every agent of the basic MAS has to be adapted.

6. ***Send* derived rules to the agents:** Finally, the EIA transfers newly created rules to the agents the next time it can communicate with them. From the moment the agent has stored a new rule in its rule base, the rule will be incorporated into the agents' decision mechanism (usually defined by simple local rules), which provides them in specific situations with a limited capability to "look into the future". However, exception rules can be ignored by the agents, and will be ignored, when they are not useful anymore.

Because as a result all problem solving decisions are still locally made by the agents, the latter will continue to work, even if the EIA fails. Thus, we obtain the benefits of a central control but avoid the problems that are usually associated with it. For a more formal definition of the advisor's actions as well as an instantiation of this general architecture to the domain of PDPs, we again refer to [20].

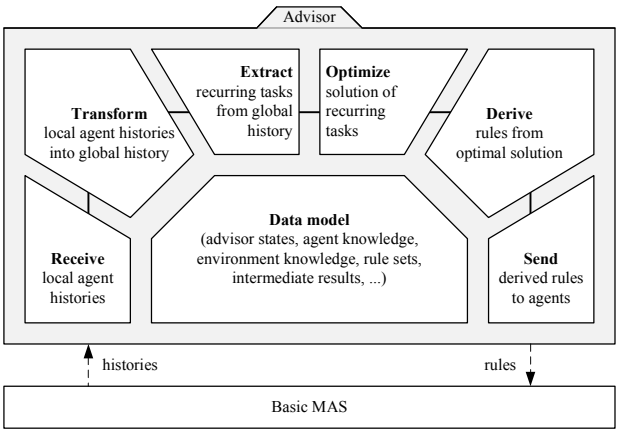

Figure 1: Functional architecture of an advisor

3. SOLVING PDPS BY ENVIRONMENT-MEDIATED MULTI-AGENT SYSTEMS

Due to the high complexity and dynamism of today's PDPs paired with the need for more flexibility, openness, and autonomy of solutions, in recent years self-organizing emergent systems became a promising solution technology. However, their poor performance in some cases is a crucial fact that hinders their acceptance by industry.

3.1 Pickup and Delivery Problems

The PDP is an omni-present problem that is faced each day by thousands of companies and organizations engaged in the transportation of goods. Roughly spoken, it concerns the service of a set of customers in a given time period by a set of vehicles, which are located in one (or more) depots and perform their movements by using an appropriate road network (even though PDPs can be extended to air, rail, water, and pipeline networks as well). A solution of a PDP calls for the determination of a set of routes, each performed by a single vehicle that starts and ends at its own depot, such that all requirements of the customers are fulfilled, all the operational constraints are satisfied, and one or more global optimization objectives are reached. In contrast to static PDPs [1], where all necessary information (number of

vehicles, transportation requests, ...) is available a priori, in dynamic PDPs [2], which we are interested in, at least some information is revealed during the period of time in which operations take place. In a PDP with time windows (PDPTW), additionally a time interval is associated with each customer, in which the goods must be pickup up respectively delivered. Dynamic PDPTWs appear in various application domains, such as courier services, manufacturing control, aircraft sharing, distribution of heating oil, or emergency vehicle dispatching.

3.2 Environment-Mediated MAS Solutions

A MAS solution to a PDP usually consists of at least two types of agents: vehicle agents, representing trucks or automated guided vehicles (AGVs), and station agents, representing pickup/delivery customers. In general, there exist a couple of decentralized coordination mechanisms to solve PDPs by a MAS, i. e. to assign the tasks (transportation requests) to vehicle agents, e. g. by using market-based coordination such as auctions [16] or the Contract Net Protocol [3], token-based coordination [7], immunity-based coordination [12], field-based coordination (FBC) [21], or pollination-inspired coordination (PIC) [9]. Since environment-mediated MAS solutions, as exemplified by the two latter coordination mechanisms, have shown to result in manageable solutions with very adaptable basic qualities, we focus on this class of self-organizing emergent MASs in the following. Although both mechanism have a couple of concepts in common, there also exist conceptual differences.

3.2.1 *Field-based Coordination*

The coordination by means of computational fields is a well-studied domain in MAS [14]. The basic idea of field-based solutions to dynamic PDPs is to let each idle vehicle agent follow the gradient of a field that guides it toward a task that has to be executed. The vehicle agents continuously reconsider the situation in the environment and task assignment is delayed until the execution of the task starts, which benefits the flexibility of the system.

Field-based task assignment (FiTA) is achieved by the interaction between vehicle agents and station agents. Both vehicle and station agents emit fields in the local environment. Station fields attract idle vehicle agents. However, to avoid multiple vehicle agents driving toward the same station agent, the former emit repulsive fields. Vehicle agents combine perceived fields and follow the gradient of the combined fields, that guide them toward pickup locations of requests. Fields have a certain range and contain information about the source agent. Vehicle fields have a fixed range, while the range of station fields is variable and depends on the actual priority of the tasks. Fields are refreshed at regular times, according to a predefined refresh rate.

When a vehicle agent perceives fields, it stores the data contained in these fields in a field-cache, which consists of a number of cache entries. Each cache entry contains the identity of the perceived field, the most recent data contained in that field, and a freshness, which is a measure of how up-to-date the cached data is. By a combination of the perceived fields, each vehicle agent calculates for each possible direction to move a coordination-field to decide in which direction to drive (see [21] for a detailed description of the calculation of the coordination-field), whereby the lower the freshness of a cache-entry, the lower the influence of the associated field on the coordination-field. A vehicle agent follows the coordination-field in the direction of the smallest value, which can be considered as following the gradient of the coordination-field downhill.

3.2.2 *Pollination-inspired Coordination*

PIC is initially inspired by the pollination of flowers by honey bees and based on the IBC model [10]. In the same way as flowers emit synomones (chemical compounds) into the air in order to attract bees for the pickup/delivery of pollen grains, station agents emit their digital counterparts, which contain information on the identity of the emitter, in order to attract vehicle agents for transportation. To avoid multiple vehicle agents driving toward the same station agent, vehicle agents emit digital pheromones (another chemical compound), indicating their current target, which keeps following agents from driving to the same target. Similarly, if the request of a station agent has been already fulfilled by a vehicle agent, the former emits digital allomones (another chemical compound), in order to keep further vehicle agents from visiting.

Every emitted digital infochemical holds individual information and has a concentration value associated with it, which decreases as a function of time and distance. In other words, an infochemical is propagated in a certain range as long as the concentration is higher than a given threshold and will finally evaporate, if it is not refreshed. The infochemicals are stored at each location within the propagation range. A vehicle agent combines all perceived infochemicals stored at its current location and drives in the direction with the highest utility, which is usually the one with the highest synomone concentration, assumed that no allomone or pheromone reduces the utility of following this synomone to zero (see [9] for a detailed description of the calculation of the utility).

Similar to FiTA, final task assignment is delayed until a vehicle agent actually reaches a station agent and picks up the goods. This allows vehicle agents to adapt the assignment of tasks while they drive toward goods and thus to cope with changing circumstances that arise on their own. In contrast to FiTA, the communication between the agents in PIC is indirect by means of the environment storing the infochemicals, whereas in FiTA the agents themselves store the perceived fields. Also, in FiTA the vehicle agent follow the smallest value of a coordination-field, whereas in PIC the agents follow the highest utility value. Nonetheless, both environment-mediated coordination mechanisms suffer from the same inefficiencies.

3.3 Inefficiencies

Based on our experiences with environment-mediated MAS solutions based on FiTA (see [21]) respectively PIC (see [9]) to PDPs, we have extracted and analyzed several types of situations, in which local decisions of single agents lead to inefficiencies in the global solution. The reasons for these suboptimal local decisions are threefold: (1) the greediness of the agents, (2) the reactiveness of the agents, and (3) the absence of global knowledge. Thus, most of these situations may also appear in other, not environment-mediated, self-organizing emergent MAS solutions, in which the agents are greedy and/or reactive, and in which the communication between the agents does not provide every agent with global knowledge, respectively. Please note that due to the

nature of emergence the extraction of all possible situations exhibiting inefficiencies in self-organizing emergent solutions in general is impossible.

In order to determine the efficiency of a solution and with it the inefficiencies, we were interested in two different quality measures: (1) total travel costs (TTC), i. e. the distance the vehicle agents traveled in order to fulfill all transportation requests, and (2) violation of time windows, i. e. the time elapsed after the deadline of a transportation request.

3.3.1 Requests Handled by Inappropriate Agents

Due to their greediness, vehicle agents try to serve any request right upon the perception of the corresponding synomones (respectively attracting field in FiTA), assumed that no allomones or pheromones (respectively repelling fields in FiTA) hinder the agents. Because the agents in particular cannot reject or defer a request, usually, the idle vehicle agent, which is closest to a pickup request, will try to serve it, disregarding other requests in the vicinity and without regard of requests that might appear at the agent's initial location or its vicinity in the near future (see example later in Section 5). This results in unnecessary long routes and thus in an increase of total travel costs, as other vehicle agents might be in a much better position to serve the request. It also results in a violation of time windows, as another vehicle agent has to approach from a location more far off to serve the likely future request.

3.3.2 Attraction of Multiple Vehicle Agents

Due to their greediness, in many cases, two or more vehicle agents may initially start to approach the same station agent to serve a pickup request. Even though all vehicle agents leave a pheromone trail respectively emit a repelling field that keeps other agents from approaching to the same target, this trail (field) is restricted to a limited area around the emitting agent and can therefore only be perceived by other vehicle agents that are close by. If the vehicle agents approach from different directions to the same station agent, the trails (fields) have obviously only a late effect, which results in unnecessary movements and consequently increases total travel costs as well as violates time windows.

3.3.3 Overcrowding of Pathways

Due to their greediness, in small environments or environments including bottlenecks in regard of certain pathways, the announcement of a huge amount of transportation requests within a short period of time may instigate a huge number of vehicle agents to leave the depot immediately. In particular if the amount of requests exceeds the amount of vehicle agents, all agents will start to follow a different synomone (attracting field), because the pheromones (repelling fields) emitted by each agent indicate different targets. Due to the service times at pickup stations, which depend on the load sizes of the requests, as well as the waiting times of the following agents, congestions emerge and the vehicle agents stand in each others way, which results in the violation of many time windows.

3.3.4 Oscillation of Vehicle Agents

Due to their greediness, a vehicle agent may repeatedly shuttle between the depot and pickup stations, if the transportation requests appear at these stations with an adverse interval (see example later in Section 5). In other words, the period of time between the occurrence of the transportation requests is higher than the period of time the vehicle agents require to move from a delivery station back to the depot. This results in an increase of total travel costs and sometimes to a violation of time windows.

3.3.5 No Consideration of Stochastic/Static Requests

Due to their reactiveness, vehicle agents will only start to move to a station agent, if the latter has emitted appropriate synomones (attracting field). However, in stochastic PDPs or PDPs with a mixture of static and dynamic requests, a certain amount of requests have to be taken into account that are already known a priori. In particular for tasks with tight time windows, this results in a violation of the latter, because the vehicle agents first have to spend the time for moving to a station agent demanding the service of a stochastic/static request, which they could have saved, if they would have started prior to the perception of the corresponding synomones (attracting field).

3.3.6 Undetected Requests

Due to their reactiveness, vehicle agents situated at the depot that do not perceive any appropriate synomones (attracting fields), e. g. due to a too short propagation range, will remain in the depot, even if appropriate synomones (fields) could be found in the immediate vicinity of the depot. The same holds for vehicle agents situated out of the depot that do not perceive any appropriate synomones (attracting fields), which in such cases will (after a given period of time) return to the depot. Thus, requests at worst may remain undetected, or will be detected at some remote period, which increases travel costs and violates time windows.

3.3.7 Unserved Requests

Due to their reactiveness, vehicle agents following a synomone (attracting field) to a station agent in order to serve a pickup request, may decide to follow another perceived synomone (attracting field), assumed that its utility is higher respectively computational-field value is smaller. Thus, due to this reconsideration of the environment, the original request may remain unserved, in particular if other vehicle agents have already stopped following the synomones (attracting field) to the original request, as they have perceived the pheromones (repelling field) of the switching agent. This not only results in an increase of total travel costs, but also in a violation of time windows.

4. EXCEPTION RULES

Because all identified types of inefficiencies emerge from the local behavior of the vehicle agents, in this section we identify and classify different exception rules by which an EIA can adapt the local behavior of vehicle agents in order to improve the efficiency of the global solution. Therefore, the underlying coordination principles of the used coordination mechanism have to be taken into account to find the best way to formulate the exception rules.

In general, for environment-mediated MASs we distinguish three classes of exception rules employable by an EIA (see Figure 2): task-triggered rules, time-triggered rules, and neighborhood-triggered rules. As the name implies, *task-triggered rules* become activated in the rule base of an agent by the perception of a certain task, e. g. by the perception of a certain synomone (attracting field) referring to a trans-

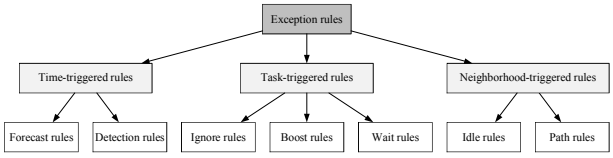

Figure 2: Exception rules to adapt the local behavior of agents in environment-mediated MASs

portation request. In other words, a task must be present first in order to activate such an exception rule. By contrast, *time-triggered rules* do not require the presence of tasks but become activated due to the passing of a certain point in time or after a certain period of time. Apparently, these rules require the agents to have a common notion of time, e. g. the time measurement may start at the beginning of a day or the beginning of a run instance. Finally, *neighborhood-triggered rules* become activated by the behavior of the other agents acting in the same environment, i. e. by the perception of pheromones (repelling fields).

Task-triggered exception rules can be further distinguished into the following three types:

- **Ignore rules:** This rule type forces an agent to ignore for a given period of time a perceived infochemical (field) that is sufficiently similar to an abstracted infochemical (field). Thus, the utility of following this infochemical is set to zero respectively the computational-field value of this field is set to ∞. When the period of time has passed, the rule is no longer applied and the perceived infochemical (field) is evaluated as it would normally have been. Ignore rules are very powerful and may be used by the EIA to deter an inappropriate or superfluous agent from serving a task (see 3.3.1 and 3.3.2, respectively), or to prevent the agent from oscillating (see 3.3.4).

- **Boost rules:** This rule type forces an agent to boost for a given period of time and by a given percentage the utility of a perceived infochemical resp. the value of a gradient field that is sufficiently similar to an abstracted infochemical (field). This increases the chance of following this infochemical (field) compared to other ones. If the percentage is set to a value lower than 100%, the infochemical utility resp. gradient field value will be lowered accordingly. Consequently, ignore rules may be considered as a subclass of boost rules, in which the utility of an infochemical is boosted by 0% respectively the value of a gradient field is boosted by ∞. When the period of time has passed, the rule is no longer applied and the perceived infochemical (field) is evaluated as it would normally have been. Boost rules may be used by the EIA to guide an agent to a more optimal task in the vicinity (see 3.3.1) or to prevent an agent from changing its mind too often (see 3.3.7).

- **Wait rules:** This rule type forces an agent to wait for the perception of an infochemical (field) that is sufficiently similar to an abstracted infochemical (field). In other words, the agent ignores all infochemicals (fields) that are not sufficiently similar to the abstracted infochemical (field). However, the rule does not induce the agent to follow the infochemical (field) it has waited for, but only to wait upon its perception. At this time the agent evaluates again, which infochemical (field) to follow. In addition, a deadline has to be given, as otherwise the agent will wait until the end of the run instance and will remain unproductive, if the task the abstracted infochemical (field) refers to is not part of the run instance. Wait rules may be used by the EIA to deter an inappropriate agent from serving a couple of tasks (see 3.3.1) or to prevent an agent from leaving a location, e. g. the depot, too early (see 3.3.4).

In contrast to the EIA, an agent has no notion of a task. Thus, an abstracted infochemical (field) has to be constructed by the EIA as an extraction from an identified task. The similarity between a perceived infochemical (field) and an abstracted infochemical (field) can be determined by an appropriate distance or similarity function.

Time-triggered rules[1], as the second class of exception rules, can be distinguished into two types:

- **Forecast rules:** This rule type forces an agent to move to an abstracted location or area, as soon as a given point in time is reached. However, the rule does not induce the agent to follow a specific route or to serve a specific task, which is still up to the agent's decision. As soon as the agent has arrived on the abstracted location, it has to wait for a given period of time. Thus, forecast rules may be used by the EIA to send an agent into an area, where the occurrence of a stochastic/static request is very likely at a specific time (see 3.3.5). Without the waiting constraint, the effect of the rule will get lost, if the dedicated request has not appeared yet, because the agent might then decide to serve another request instead.

- **Detection rules:** This rule type forces an agent to move according to a given strategy, if the agent does not perceive any useful infochemical (field) for a given period of time. A strategy may either include a couple of abstracted locations the agent has to visit or describe a certain movement pattern. Detection rules may be used by the EIA to let the agent search for yet undetected requests (see 3.3.6).

Finally, neighborhood-triggered rules can be distinguished into two types as well:

- **Idle rules:** This rule type forces an agent to remain idle for a given period of time, if the perceived amount of infochemicals, e. g. pheromones, respectively the perceived strength of repelling fields exceeds a given threshold. Idle rules thus can be used by the EIA to prevent an overcrowding of pathways or to limit the number of agents currently acting in the environment, assumed that the agent is still in the depot (see 3.3.3).

- **Path rules:** This rule type forces an agent to move according to a given strategy, which may again include

[1]Because the basic movement capabilities of agents acting based on PIC or FiTA essentially require the presence of infochemicals (fields) stored at locations, time-triggered rules additionally require the presence of a global environment map stored in each agents' data model, along with an appropriate routing algorithm that has to be incorporated by the agents. FiTA already incorporates such a map.

a couple of abstracted locations the agent has to visit successively or describe a certain movement pattern, if the amount of perceived infochemicals respectively a field strength exceeds a given threshold for a given period of time. Path rules may be used by the EIA in similar cases as idle rules (see 3.3.3), however, these rules are better suited for situations in which the agents are out of the depot. Thus, instead of idling and possibly blocking other agents, the agent may be advised to use an alternative path in order to avoid congestions.

Although possible, a combination of different rule types has to be used carefully. For example, advising a vehicle agent to search for undetected requests by a detection rule may in turn result in an attraction of multiple vehicle agents to the same station agent (inefficiency 3.3.2). Similarly, even though some rule types are more or less conflict free taken by itself, e. g. ignore rules, other rule types, in particular time-triggered rules, or a combination of different rule types may require some kind of deliberation. On the other hand, by a purposeful combination of different rule types the advice for an agent can be more specific. For example, letting an agent ignore a certain infochemical (field) and boost the utility/gradient value of another one, the reaction of an agent may be anticipated very well.

5. CONCEPTUAL EVALUATION

In [20], we have experimentally demonstrated the flexibility of an EIA, as it is able to identify crafted, random, and changing task patterns in arbitrary PDPs, in which the tasks are handled by inappropriate agents (see 3.3.1). We showed that the adaptation of these agents by ignore rules yields efficiency improvements of up to 30%. In this section, we focus on similar situations, however, in which the adaptation by ignore rules yields no improvements respectively actually worsens the global solution, and thus require the employment of other task-triggered rules. Due to the limited space we cannot describe the conceptual evaluation of time-triggered and neighborhood-triggered rules.

For the conceptual evaluation, we take an automated transportation system employing multiple AGVs for automating logistics services in warehouses or manufacturing facilities as a starting point. Furthermore, we assume that the EIA has already extracted from the local agent histories a common scenario as depicted in Figure 3. The depot is represented by the black circle with $ID = 0$, while the black circles with IDs 1 to 6 represent pickup/delivery stations. Vehicle agents can move along a connection between two locations generating travel costs of 1 for a horizontal resp. vertical move and $\sqrt{2}$ for a diagonal move. Furthermore, we assume that the EIA has also identified that the vehicle agent $\mathcal{A}g_1$, initially located at the depot and having a transportation capacity of 10 goods, is in charge of executing the three tasks $ta_1 = (3, 4, 10, 20, \infty)$[2], $ta_2 = (1, 2, 10, 20, \infty)$, and $ta_3 = (5, 6, 10, 20, \infty)$ (nearly) every run instance, while other agents are in charge of executing other tasks not mentioned here. The dashed circles indicate the propagation range of the synomones resp. the attracting fields of the three tasks. Apparently, in terms of total travel costs, the optimal execution order for $\mathcal{A}g_1$ would be ta_2, ta_1, ta_3, yielding TTC of 26,38. However, without any adaptation of the local behavior, an $\mathcal{A}g_1$ based on FiTA or PIC first executes ta_1, because the pickup station is the closest to the depot and thus provides the highest utility resp. lowest computational-field value (see 3.3.1), before executing ta_3 and finally ta_2, yielding TTC of 29,56 (+12,05%).

[2]This notation of a task means that 10 goods have to be transported from station agent 3 to 4. The tasks becomes available at time step 20 and has no deadline specified.

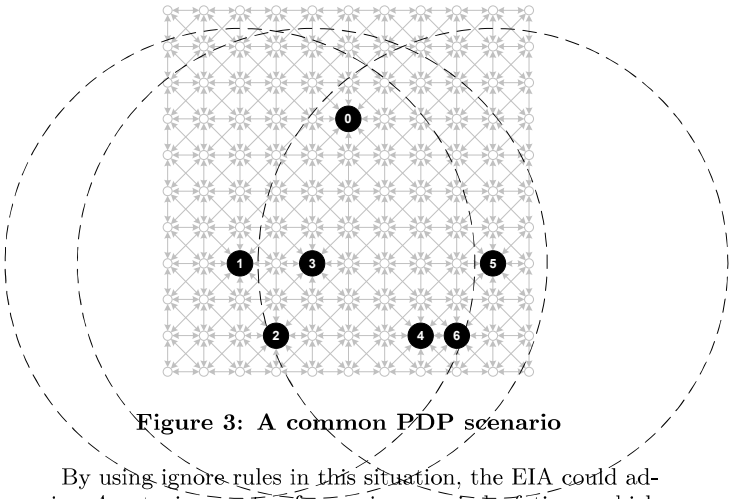

Figure 3: A common PDP scenario

By using ignore rules in this situation, the EIA could advice Ag_1 to ignore ta_1 for a given period of time, which consequently would prompt Ag_1 to execute ta_2 first, as required. However, if we assume that Ag_1 still ignores ta_1, it would execute next ta_3 and then return to the depot or execute ta_1, if the given period of time for ignoring ta_1 is already over. Thus, the ignore rule would worsen the solution by at least 22,76%. By using a boost rule instead, the EIA advices the agent to boost the utility (field value) of ta_2 by a given percentage, which prompts Ag_1 to execute ta_2 first, ta_1 second, and finally ta_3, which represents an improvement of 12,05% compared to the regular solution.

If the EIA in contrast identifies task patterns that do not become available at the same time, e. g. $ta_1 = (3, 4, 10, 20, \infty)$, $ta_2 = (1, 2, 10, 50, \infty)$, and $ta_3 = (5, 6, 10, 80, \infty)$, without any adaptation Ag_1 will execute these tasks in the same order as they appear and in the meantime will always return to the depot (see 3.3.4), yielding TTC of 44,87. By using a wait rule, the EIA however could advices Ag_1 to wait at the depot until the perception of ta_3, and only then to start executing the tasks, still with a combination of the boost rule described above, which yields an overall improvement of 70,09% compared to the regular solution.

This evaluation demonstrates how the local behavior of a self-organizing agent can be adopted by exception rules, in order to improve the global solution.

6. CONCLUSIONS AND OUTLOOK

Based on our experiences with environment-mediated MAS solutions based on PIC and FiTA to PDPs, in this paper we have extracted and analyzed several types of situations, in which local decisions of single agents lead to inefficiencies in the global solution of the system. Usually, not all of these situations appear every time in every solution, as parameter settings have a significant effect. However, some of these situations may appear in other self-organizing emergent MAS solutions as well, if the used coordination mechanism employs greedy or reactive agents having no global knowledge.

We furthermore have presented an approach, how to au-

tonomously adapt the local behavior of misbehaving agents in these situations by exception rules, in order to improve the global solution of the self-organizing emergent system, regarding relevant challenges and constraints that have to be respected in this process. We have demonstrated the potential of two of these exception rules by a conceptual evaluation. Because in PIC-based and FiTA-based solutions the way the agents react to their environment is based on the perception and evaluation of infochemicals resp. gradient fields, task-triggered exception rules that influence the evaluation process prove to be a very good starting point for the adaptation of environment-mediated MAS solutions. However, in particular in regard of time windows, we expect time-triggered rules to achieve high improvements as well.

Our next steps consist of building a joint simulation tool, in order to evaluate the adaptation by exception rules for various other coordination mechanisms apart from PIC and FiTA as well as other application domains apart from PDPs. In particular for resource (such as energy, water, or gas) distributions networks we expect the application of the EIA approach to be very promising. Furthermore, the evaluation showed that due to the diversity of exception rules, an EIA has to be capable of simulating the effects of an exception rule (set) first, to identify the most appropriate rule (set) to apply in a specific situation, before sending the rule (set) to the basic agents. However, due to the characteristics of emergent systems a single simulation might be inadequate, which might require several simulations in a restricted time.

Although the EIA represents a promising approach to combine the principles of self-organizing and self-adaptive systems, some essential questions for future research remain: What issues arise due to the fact that local autonomy of agents is in part overruled by exception rules? How to guarantee that the adaptation of the local behavior is not counterproductive and possibly worsens the global solution in awkward situations? What about graceful degradation, if one or more adapted agents fail? Will scalability in terms of millions of agents be an issue for real-world application domains, or can we assume to split such big problems into several subproblems only requiring some hundreds of agents to participate at most? Assumed that an optimal solution to a problem can be calculated, how close can we get to this solution by an adapted self-organizing emergent system?

7. REFERENCES

[1] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static Pickup and Delivery Problems: A Classification Scheme and Survey. *TOP*, 15(1):1–31, 2007.

[2] G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic Pickup and Delivery Problems. *EUR J OPER RES*, 202(1):8–15, 2010.

[3] N. Boucké, D. Weyns, T. Holvoet, and K. Mertens. Decentralized Allocation of Tasks with Delayed Commencement. In *Proc. of EUMAS 2004*, pages 57–68, 2004.

[4] 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 ISoLA 2006*, pages 200–206, 2006.

[5] B. H. C. Cheng et al. Software Engineering for Self-Adaptive Systems: A Research Road Map. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 1–26, 2009.

[6] T. De Wolf and T. Holvoet. Emergence Versus Self-Organisation: Different Concepts but Promising When Combined. In *Engineering Self-Organising Systems*, volume 3464 of *LNCS*, pages 1–15, 2004.

[7] A. Farinelli, L. Iocchi, D. Nardi, and V. A. Ziparo. Task Assignment with Dynamic Perception and Constrained Tasks in a Multi-Robot System. In *Proc. of ICRA 2005*, pages 1523–1528, 2005.

[8] K. Fischer. Cooperative Transportation Scheduling: An Application Domain for DAI. *Applied Artificial Intelligence*, 10(1):1–34, 1996.

[9] H. Kasinger, B. Bauer, and J. Denzinger. The Meaning of Semiochemicals to the Design of Self-Organizing Systems. In *Proc. of SASO 2008*, pages 139–148. IEEE CS, 2008.

[10] H. Kasinger, B. Bauer, and J. Denzinger. Design Pattern for Self-Organizing Emergent Systems Based on Digital Infochemicals. In *Proc. of EASe 2009*, pages 45–55. IEEE CS, 2009.

[11] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[12] H. Y. Lau, V. W. Wong, and I. S. Lee. Immunity-based Autonomous Guided Vehicles Control. *Applied Soft Computing*, 7(1):41–57, 2007.

[13] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case Studies for Self-organization in Computer Science. *Journal of Systems Architecture: the EUROMICRO Journal*, 52(8):443–460, 2006.

[14] M. Mamei and F. Zambonelli. *Field-Based Coordination for Pervasive Multiagent Systems.* Springer, 2006.

[15] M. Mes, M. van der Heijden, and A. van Harten. Comparison of Agent-based Scheduling to Look-ahead Heuristics for Real-time Transportation problems. *EUR J OPER RES*, 181(1):59–75, 2007.

[16] M. Mes, M. van der Heijden, and J. van Hillegersberg. Design Choices for Agent-based Control of AGVs in the Dough Making Process. *Decision Support Systems*, 44(4):983–999, 2008.

[17] M. W. P. Savelsbergh and M. Sol. The General Pickup and Delivery Problem. *Transport Sci*, 29:17–29, 1995.

[18] R. Schumann, A. D. Lattner, and I. J. Timm. Management-by-Exception - A Modern Approach to Managing Self-Organizing Systems. *Communications of SIWN*, 4:168–172, 2008.

[19] G. D. M. Serugendo, M.-P. Gleizes, and A. Karageorgos. Self-Organisation and Emergence in MAS: An Overview. *Informatica*, 30(1):45–54, 2006.

[20] J.-P. Steghöfer, J. Denzinger, H. Kasinger, and B. Bauer. Improving the Efficiency of Self-Organizing Emergent Systems by an Advisor. In *Proc. of EASe 2010*, pages 63–72. IEEE CS, 2010.

[21] D. Weyns, N. Boucké, and T. Holvoet. A Field-based Versus a Protocol-based Approach for Adaptive Task Assignment. *JAAMAS*, 17(2):288–319, 2008.

[22] D. Weyns, S. A. Brueckner, and Y. Demazeau, editors. *Engineering Environment-Mediated Multi-Agent Systems.* Springer, 2008.