An advisor concept for distributed self-organizing systems acting in highly connected environments

Philipp Grosselfinger, Jorg Denzinger, Bernhard Bauer

Angaben zur Veröffentlichung / Publication details:

Grosselfinger, Philipp, Jorg Denzinger, and Bernhard Bauer. 2013. "An advisor concept for distributed self-organizing systems acting in highly connected environments." In *Proceedings of the 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems, 9-13 September 2013, Philadelphia, PA, USA*, edited by Peppo Valetto and Ozalp Babaoglu, 121–30. Los Alamitos, CA: IEEE. https://doi.org/10.1109/saso.2013.15.

An advisor concept for distributed self-organizing systems acting in highly connected environments

Philipp Grosselfinger
Department of Computer Science
University of Augsburg,
86135 Augsburg, Germany
Email: philipp.grosselfinger@gmail.com

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

Bernhard Bauer
Department of Computer Science
University of Augsburg,
86135 Augsburg, Germany
Email: bauer@informatik.uni-augsburg.de

***Abstract*—We present an extension to the concept of an advisor for distributed self-organizing systems that allows to improve the efficiency of such systems that act in environments where the actions of a system component have an influence on the whole environment or large parts of it. An advisor periodically reviews the history of the system and identifies recurring tasks that the system did not perform well. For these tasks it then computes exception rules for the agents that indicate how to perform these tasks better. These rules are communicated to the agents when communication is possible and are from then on used if they are triggered. To deal with highly connected environments so-called group exceptions are needed that consist of individual rules for several agents and that require the agents to signal to the other agents that in their local view the group rules should be triggered.**

We instantiated this group advisor concept to a self-organizing system for controlling water distribution networks that is based on digital infochemical coordination. In our experiments, using the group advisor resulted in improvements in energy use between 1 and 21 percent, with an average savings of $ 5,000 per day for randomly generated water demand behaviors for the real network of a small North-American city.

I. Introduction

Dynamic distributed optimization problems, like controlling a group of pickup-and-delivery vehicles, controlling the power plants in a power grid, or controlling the pumps in a water distribution network or a pipeline, are a very important problem class in todays world. While until recently these problems have been mostly dealt with by human controllers, perhaps with some help by static optimization software (if enough time was available), in the last few years self-organizing and self-adapting systems have shown to possess many useful properties for the solution of these optimization problems. Among these useful properties are scalability, flexibility and robustness.

Concepts like Digital Infochemical Coordination (DIC), see [8], or field-based coordination, see [10], have shown these properties in a number of applications. Unfortunately, from the economical perspective, these approaches all have one big problem, namely the quality of the produced emergent solutions. While due to the dynamic nature of the problems it can not be expected that optimal solutions are produced (this would require the ability to look into the future), the emergent solutions can be substantially worse than the optimal solutions ([6] reports instances that are 3 to 4 times worse than known very good solutions).

Recently, in [14] the concept of a so-called Efficiency Improvement Advisor (EIA) was presented to deal with the inefficiency of basic self-organizing systems. As the term "advisor" suggests, the EIA is not directly part of the basic self-organizing system and its operations in solving instances of dynamic optimization problems. Instead, the EIA periodically collects information from all of the components of the self-organizing system, the *agents*, about their history, uses this information to construct a global history of the system, tries to detect recurring tasks for the system, determines if the system solves these tasks efficiently, and if not, devises advice in the form of exception rules for the agents that is aimed at improving the efficiency of the system when these recurring tasks occur in the future. [14] presented one type of exception rules, that aimed at keeping an agent away from a particular task, while [15] introduced pro-active exception rules that forced agents to do a particular task. Both works showed that the EIA leads to efficiency improvements, if enough of the tasks in a working period (like a day) are recurring over several periods.

So far, in all of the applications of the EIA, both the detection of a recurring task and the performing of a task was done locally, i.e. a single agent was capable to determine if an exception rule should be applied and to perform the action(s) necessary to complete the detected task. Unfortunately, there are environments, for example the hydraulic system in a pipeline, where the actions of an agent do not just influence the environment locally but globally and where therefore an exception needs to be triggered based on information from more than one agent (all agents in the worst case).

In order to deal with this problem of highly connected environments, in this paper we present an extension to the EIA concept that we call Group Efficiency Improvement Advisor (GEIA). The general work cycle of the GEIA is similar to EIA, but if GEIA detects recurring situation sequences that the basic self-organizing system does not handle well it creates a group of exception rules for a group of agents that aim at improving the behavior of these agents for the situation sequences (or similar sequences). While each of these exception rules is for a specific agent, so that the modifications to agents are mostly similar to EIA, triggering such a rule obviously requires

some kind of compacted global view of the system. In order to keep the robustness of the basic system, we achieve this compacted view by having all the agents of such an exception rule group periodically send a "rule triggered" signal while the local conditions for the group exception rule are fulfilled. And if an agent receives these signals from all other agents (and has its local condition fulfilled), it executes the action sequence determined by the triggered exception rule instead of its usual reactive behavior until either the sequence is finished or at least one agent is not sending the rule triggered signal anymore.

We instantiated the GEIA concept for the self-organizing system for controlling water distribution networks (WDNs) based on DIC presented in [5]. Our experiments with the resulting system and several different WDNs showed improvements in the energy required to ran the networks under different randomly created demand run behaviors (with recurring constant demand periods) between 1 and 10 percent. Selected demand behaviors were even improved by 10 to 21 percent.

II. Basic definitions

In this section, we provide some basic definitions around agents, multi-agent systems and the kind of dynamic distributed optimization problems we are interested in. A very general definition of an agent $\mathcal{A}g$ that we use to structure the description of our agents is as a 4-tuple $\mathcal{A}g$ = (Sit, Act, Dat, f_{Ag}), where Sit is the set of different situations the agent can perceive, Act is the set of actions the agent can perform, Dat the set of possible value combinations of the agent's internal data areas and $f_{Ag} : Sit \times Dat \rightarrow Act$ is $\mathcal{A}g$'s decision function, describing how it selects an action based on its current situation and the current value of its internal data areas. A multi-agent system (MAS) is a set of agents $A = \{\mathcal{A}g_1, \mathcal{A}g_2, \ldots, \mathcal{A}g_n\}$ that are acting in a shared environment $\mathcal{E}nv$.

The class of problems that we are interested in tries to solve tasks from a set T that are announced in $\mathcal{E}nv$ during a time interval $Time$. A sequence ((ta_1, t_1), (ta_2, t_2), …, (ta_m, t_m)), with $ta_i \in T$, $t_i \in Time$ and $t_i \leq t_{i+1}$ is called a *run instance* where the t_i are the times at which the tasks ta_i appear in $\mathcal{E}nv$. Typically, a problem consists of a sequence of run instances. A sequence of run instances of length k is then $(ri_1, ri_2, \ldots, ri_k)$. A solution sol generated by a group of agents A for a run instance is again a sequence sol = ((ta'_1, $\mathcal{A}g'_1$, t'_1), (ta'_2, $\mathcal{A}g'_2$, t'_2), …, (ta'_m, $\mathcal{A}g'_m$, t'_m)) where $ta'_i \in \{ta_1, \ldots, ta_m\}$, $ta'_i \neq ta'_j$ for all $i \neq j$, $\mathcal{A}g'_i \in A$, $t'_i \leq t'_{i+1}$, and $t'_i \in Time$.

To determine the efficiency of A in solving a run instance, we associate with each solution sol a quality measure $qual(sol)$ that, depending on the problem, needs to be either maximized or minimized. Since usually the agents need to start fulfilling tasks before all tasks of a run instance are known to them, it is not possible to guarantee optimality for emergent solutions.

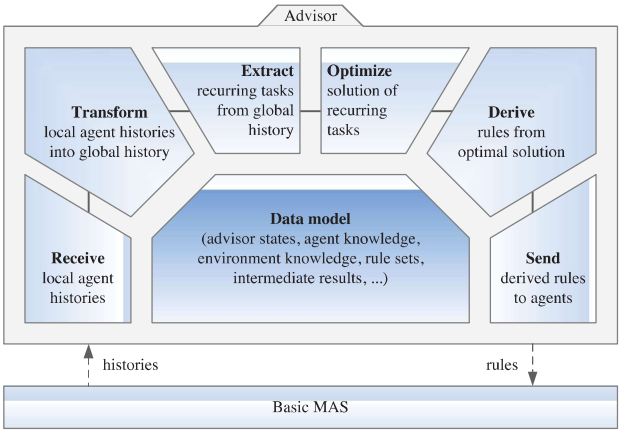

Fig. 1. Functional architecture of EIA

III. The EIA concept

In this section, we will first present the requirements on a distributed system in order to be able to use an EIA. Then we will go through the general working cycle of the EIA and finally concentrate on how the agents of the basic system need to be modified to use the advice generated by the EIA.

There are three basic requirements on a distributed self-organizing system that need to be fulfilled in order for the EIA to be useful in improving the efficiency of the generated solutions over time.

1) Every agent is capable to send its local history to the EIA at least once during or after a run instance is performed.
2) Every agent's decision function should be extendable to handle exception rules, and it should be possible to store these rules in an internal data area of the agent.
3) There must be quite a number of (similar) recurring tasks in most run instances.

The first two requirements are easy to fulfill, even in systems where single agents cannot perceive the whole environment. And in many dynamic distributed optimization problems of relevance, there are quite a number of tasks that keep recurring.

The general working cycle of the EIA is depicted in Figure 1. For many of the different steps in the working cycle there are many possible ways how these steps can be realized. In the following we will provide a high-level description of these steps, including the necessary arguments (using the definitions from the last section) to perform a step and the results of each of the steps (see also [14]).

receive($\mathcal{A}g_i$,((s_i^1,d_i^1,a_i^1),…,(s_i^o,d_i^o,a_i^o))) collects the local history H_i for each agent $\mathcal{A}g_i$, when $\mathcal{A}g_i$ is able to communicate, while A performs a run instance. H_i = ((s_i^1, d_i^1, a_i^1),…,(s_i^o, d_i^o, a_i^o)), with $s_i^l \in Sit_i$, $d_i^l \in Dat_i$, $a_i^l \in Act_i$, is the history of $\mathcal{A}g_i$ (described by the situations the agent was in and the values of its internal data areas) since the sequence of run instances started (or a sliding window of instances).

transform(H_1,…,H_n) creates the global history $GHist$ of the whole system out of the received histories from all agents. $GHist$ essentially contains the sequence of run in-

stances (ri_1,...,ri_k) = ((ta_{11}, t_{11}),...,($ta_{m_1 1}$, $t_{m_1 1}$)),...,((ta_{1k}, t_{1k}),...,($ta_{m_k k}$, $t_{m_k k}$)) A has solved so far and the solution sol_j for each run instance ri_j that A created for it (the so-called *emergent solution*).

extract($GHist$) extracts from this history, more precisely the sequence of run instances (ri_1,...,ri_k), a sequence of recurring tasks (ta_1^{rec},...,ta_p^{rec}). In order for a task to be recurring, it, or a task that is similar enough to it according to an application dependent similarity measure sim, has to occur in most of the last few run instances A solved. The sequence of recurring tasks can be empty, in which case the EIA does nothing more until new data has arrived.

optimize(ta_1^{rec},...,ta_p^{rec}) first computes the optimal solution opt^{rec} = ((ta_1^{rec}, $\mathcal{A}g_1'^{rec}$, $t_1'^{rec}$),...,(ta_p^{rec}, $\mathcal{A}g_p'^{rec}$, $t_p'^{rec}$)), $\mathcal{A}g_j'^{rec} \in A$, $t_j'^{rec} \in Time$, assuming that ta_1^{rec},...,ta_p^{rec} were the only tasks A had to perform and they would be all known at the beginning of $Time$ (which makes this a static optimization problem). It then compares $qual(opt^{rec})$ with the quality $qual(last)$ of the last emergent solution $last$ for the tasks (ta_1^{rec},...,ta_p^{rec}) A has created. If $qual(last)/qual(opt^{rec}) > qualthresh$, then the work of $\mathcal{A}g_{EIA}$ is done until new information arrives, since A performs well. Else

derive(opt^{rec},(ta_1^{rec},...,ta_p^{rec}),$GHist$,$last$) creates for each agent $\mathcal{A}g_i$ a set R_i of exception rules, where R_i can also be empty. While there are different types of exception rules, in general each exception rule consists of a condition $cond$ on the current situation s and the current value d of the internal data areas of an agent that triggers the execution of one or several actions or prohibits the agent from executing a particular action. The exception rules are created by comparing opt^{rec} and $last$ and finding the first assignment of task to agent where the two solutions differ. Then the exception rule targets either the agent of this assignment in opt^{rec} and assigns to it the task from opt^{rec} or it targets the agent of this assignment in $last$ and prohibits it from doing the task it does in $last$. Usually, $cond$ abstracts the tasks in the situation s and the contents of d to reflect the similarity measure sim that was used in **extract**.

send($\mathcal{A}g_i$,R_i) communicates the set R_i of exception rules to an agent $\mathcal{A}g_i$ the next time communication with $\mathcal{A}g_i$ is possible.

For an agent $\mathcal{A}g_i$, its decision function $f_{\mathcal{A}g_i}$ needs to be modified in order to allow for using its set R_i of exception rules. Since we might have several rules with their conditions being fulfilled at a time (and suggesting different actions), we require the use of a conflict resolution function $cr_i : 2^{R_i} \rightarrow Act_i$. If in a situation s an agent $\mathcal{A}g_i$ has as current value of its Dat_i d, then $R_i^{s,d}$ is the set of all exception rules in R_i with fulfilled conditions. Thus, the modified decision function $f_{\mathcal{A}g_i}^{EIA}$ is defined as

$$f_{\mathcal{A}g_i}^{EIA}(s,d) = \begin{cases} f_{\mathcal{A}g_i}(s,d), & \text{if } R_i^{s,d} = \emptyset \\ cr_i(R_i^{s,d}), & \text{else} \end{cases}$$

As with the different actions of EIA, there are many possibilities for defining cr_i. For an example, please refer to [15].

IV. The group EIA concept

The EIA concept presented in the last section is centered on providing individual agents with rules for which just the individual agent can decide whether the rule should be applied. For environments in which performing a task does not influence much other areas of the environment this is sufficient to use exception rules to influence the efficiency of the system. But there are environments and tasks in them that have an impact on many if not all of the other areas of the environment than the one in which the task is fulfilled. An example are pipelines for liquids or gas where tasks usually are injecting or withdrawing liquid or gas at one point in the pipeline. This results in a change in pressure in the whole pipeline or at least all parts that are not shielded by valves from the part where the task was performed and this change in pressure also depends on the pressure and flow in all these parts. Additionally, the same pressure within a pipeline can be generated by many different setting combinations of the components of the pipeline, usually at rather different costs.

So, as for example shown in [5], controlling task fullfillment in such highly connected environments is a dynamic distributed optimization problem and can profit from applying self-organizing distributed systems. And having a variant of an advisor for such systems should improve them (in fact, it does, as can be seen in Section VI). But the advisor concept needs to be extended to allow for the need to have exception rules that need to be triggered based on information from more than the local view a single agent has on the environment and usually also more than one agent needs to have an exception rule to improve the handling of a particular task. So, this extension has to deal with groups of exception rules for groups of agents, which is why we call it the Group Efficiency Improvement Advisor (GEIA).

The GEIA goes through the same 6 steps as the EIA, in fact the first 4 steps (**receive**, **transform**, **extract** and **optimize**) are identical with respect to input and output. We only require for the **extract** step that the recurring tasks are not interrupted by non-recurring tasks between them (there might be several such sequences and each of them will be looked at by its own application of **derive**). Naturally, the step **derive** needs to be changed in order to create the groups of exception rules and determine what is needed to trigger an exception rule group. More precisely, after establishing that opt^{rec} is substantially better than the last emergent solution $last$, opt^{rec} will serve as the basis for a group exception rule gr. All agents occuring in opt^{rec} together with all agents that are known to influence these agents[1] form the group A_{gr} of agents that get their own exception rule to form the group exception rule. For an agent $\mathcal{A}g^i \in A_{gr}$, its exception rule has the form

$$cond_1^i(s_1,d_1,t_0); cond_2^i(s_2,d_2,t_1); ...; cond_p^i(s_p,d_p,t_{p-1}) \rightarrow t_1 : a_1^i, t_2 : a_2^i, ..., t_p : a_p^i$$

where $s_j \in Sit^i$, $d_j \in Dat^i$, and $a_j^i \in Act^i$, and the $cond_j^i$

[1]The set of agents that influence an agent is, naturally, application dependent. While the smaller this set the better, when in doubt considering all agents as being able to influence all other agents is always a solution.

are conditions that the current situations s_j and values d_j before time t_j have to fulfill in order for a^i_j to be executed at time t_j. The times t_j indicate offsets to the time t_0, which should be considered as a variable that is instantiated when $cond^i_1$ becomes true. So, in its general form, such an exception rule is, in fact, a sequence of exception rules that tell agent $\mathcal{A}g^i$ how to behave through the whole sequence of recurring tasks (if the whole group of agents in A_{gr} do fulfill their conditions the whole time). The conditions $cond^i_j$ are derived from the situations and values of the internal data areas that $\mathcal{A}g^i$ has at the appropriate times in opt^{rec} and usually include a local similarity measure sim^i for this particular agent. This definition of an exception rule is a generalization of a pro-active exception rule from [15], where we have a sequence of conditions and associated actions instead of just one. **send** is similar to the EIA, again.

In order to make use of the advice of the GEIA, an agent $\mathcal{A}g^i$ needs to be modified a little bit more than for EIA. Since an agent needs to inform other agents whenever according to its local view a particular exception rule should be triggered, respectively should be continued to be considered triggered, we need to include sending signals to the actions of such an agent. Such a signal $sig(\mathcal{A}g^i, gr, t)$ needs to provide the identity $\mathcal{A}g^i$ of the agent and the group exception rule gr for which $\mathcal{A}g^i$'s local conditions are fulfilled at time t. Naturally, several group exception rules might be triggered at any time and therefore several signals might have to be sent. If $Sig(\mathcal{A}g^i)$ is the set of all possible signals, then Act^i of $\mathcal{A}g^i$ needs to be modified to $Act^{i,GEIA} = Act^i \times 2^{Sig(\mathcal{A}g^i)}$. We also need to add to Dat^i a data area that holds the information that a particular group exception rule has been triggered and was selected by the conflict resolution function cr^i to be the rule that should be used. We just indicate that rule gr is the current value of that additional data area by $triggered(gr)$. Naturally, if this data area is empty, then $triggered(gr) = false$ for every rule gr. In order to avoid confusion within the whole system about what group exception rules are currently triggered, cr^i has to select the same exception rule in each agent[2].

Due to having to keep track of so many things, the definition of $f^{GEIA}_{\mathcal{A}g^i}$ contains several cases that we will go through one by one. In the following, let t be the current time, when $f^{GEIA}_{\mathcal{A}g^i}$ has to make its decision. Then

$f^{GEIA}_{\mathcal{A}g^i}(s,d) =$

- $(f_{\mathcal{A}g^i}(s,d), Sig(\mathcal{A}g^i)^t)$, if no exception rule has been triggered and $Sig(\mathcal{A}g^i)^t$ is the set of all exception rules gr of $\mathcal{A}g^i$ for which $cond^i_1(s_1, d_1, t_0) = true$ for s and d.
- $(f_{\mathcal{A}g^i}(s,d), sig(\mathcal{A}g^i, gr, t_0))$ and $triggered(gr) := true$, if $cond^i_1(s_1, d_1, t_0) = true$ and $sig(\mathcal{A}g^i, gr, t_0)$ have been received from all $\mathcal{A}g^j \in A_{gr}$, $\mathcal{A}g^j \neq \mathcal{A}g^i$ and gr was selected by cr^i.
- $(a_j, sig(\mathcal{A}g^i, gr, t_l))$, if $t_l \leq t \leq t_{l+1}$, $triggered(gr) = true$, $sig(\mathcal{A}g^i, gr, t_{l-1})$ have been received from all $\mathcal{A}g^j \in A_{gr}$, $\mathcal{A}g^j \neq \mathcal{A}g^i$ and $cond^i_j(s_l, d_l, t_{l-1}) = true$.
- $(f_{\mathcal{A}g^i}(s,d), \{\})$ and $triggered(gr) := false$, else.

Less formally, as long as no group rule has been triggered, the agent $\mathcal{A}g^i$ sends signals for all its exception rules where the initial condition $cond^i_1(s_1, d_1, t_0) = true$. If for one of these rules all other agents send their signals that this rules could be triggered before the time t_1 indicated in this rule, then the agent considers this particular rule to be triggered and starts working on the action sequence indicated by the rule (as, obviously, will do all other agents of the group for this rule). At the subsequent time steps, the agent checks if the appropriate condition on its side is fulfilled and that it still gets signals from all other agents that their local conditions are fulfilled and if all this is the case, it continues performing the appropriate actions from the rule. The moment one agent's local conditions are not fulfilled anymore, in the next step of each agent all the necessary conditions are not met anymore and the agent stops following the action sequence of its rule (and the rule gets "untriggered"). Naturally, this also means that in case of an agent "crashing", the whole system stops using any group exception rules of which this agent was part of immediately and switches to the basic self-organizing behavior if it was currently following a triggered rule.

V. Using GEIA and DIC for controlling WDNs

In this section, we first briefly present digital infochemical coordination (DIC), the concept for distributed self-organizing systems we have used to control water distribution networks (WDN), which we will present in the second sub-section. After that, we will present the instantiation of DIC for WDN from [5] and in the last sub-section we will present our instantiation of GEIA for this application.

A. Digital infochemical coordination

Digital infochemical coordination (DIC) is a generalization of pheromone-based coordination (see [2]) and aims at providing a framework for the development of emergent self-organizing systems (see [8] for an in-depth description of DIC, including a design pattern). The general idea is to achieve coordination between the agents in A using so-called digital infochemicals that are propagated through $\mathcal{E}nv$. An $\mathcal{A}g_i$ accesses all the infochemicals at its current location and bases its decisions purely on this local view of the environment. The DIC model adopts the general principles behind the communication and coordination by means of chemical stimuli between organisms in biology. These organisms may be of the same or from different species. In particular, the DIC model allows for the combination of multiple infochemicals having different semantics, dynamics, and functions, which in turn allows for more efficient self-organizing emergent solutions and solution processes.

In a system based on the DIC model, an agent of a certain type reflects a living organism of a certain species,

[2]This might not always be easy, if not all agents are in A_{gr} for each group rule gr.

able to interact indirectly by emitting and perceiving (digital) infochemicals. A (digital) infochemical ic is defined as

$$ic = (\gamma, \gamma^{thresh}, \delta, \epsilon, \psi)$$

γ is the current concentration of ic and reflects the dynamically changing concentration of diffusing biological infochemicals. γ^{thresh} is the threshold concentration of ic and reflects the behavioral threshold concentration of living organisms reacting to a specific infochemical. In other words, γ^{thresh} determines whether an agent should react to the infochemical or not. δ is the diffusion coefficient of ic and reflects the chemical diffusion coefficient that allows for a very fine-tuned propagation radius and evaporation time specific to each infochemical. ϵ is the emitter of ic and reflects a biological fact of an infochemical, namely that it reveals information on its emitter. Finally, ψ, the "payload", is individual information encapsulated by ic and reflects the biological role of an infochemical as a dynamic information carrier. The content of ψ depends on the concrete application the system using the DIC model was designed for.

B. Water distribution networks

Water distribution networks (WDNs), as they are used in our cities to bring water to both, residential and commercial customers, on a high level can be considered as a graph (V, E) of nodes $v \in V$ of various types connected by edges $e \in E$, also of various types. More precisely, the elements of V include *junctions* $ju \in Ju$, *reservoirs* $re \in Re$, and *tanks* $tk \in Ta$ (or water towers). The elements of E are *pipes* $pi \in Pi$ and we also treat *pumps* $pu \in Pu$ and *valves* $va \in Va$ as edges.

Operating a WDN is all about providing/maintaining pressure, which is called in water hydraulics *head*. If the head in the right places in the WDN is sufficient enough, a sufficient *flow* of water towards the demand sites will be present. One basic way of producing sufficient head in a place in the network is by using gravity, i.e. having these places sufficiently elevated compared to the nodes at which a certain flow of water is leaving the network. The obvious nodes in a network that should produce sufficient head are the reservoirs Re, since they are the nodes where water enters the network (in this work we will assume that each reservoir can produce unlimited amounts of water). But given that the role of a reservoir in a WDN is usually played by lakes, rivers, or even wells, having the reservoirs elevated higher than the nodes the water has to flow to is usually not an option. This is where tanks and pumps come in.

A tank tk is placed into a WDN to provide head and to act as a buffer allowing the WDN to take some time to react to changes in demand due to the water stored in tk. A tank is characterized by its elevation in the WDN, by the minimum, maximum, and current operation water level as well as its volume-height curve that provides the amount of water stored in the tank given its current water level. Since tanks do not provide unlimited amounts of water, it is necessary to get water from the reservoirs to a tank, which is accomplished using pumps. While in theory using only one pump to connect a tank with one reservoir is enough (which is why we treat a pump as an edge in the graph), in reality some kind of redundancy is required to deal with the need for maintenance and scheduled repairs/upgrades, but also with unforeseen events like a broken pipe. As a consequence, a tank usually can be replenished by several pumps, but often at different costs. These costs are measured either with regard to wear and tear of equipment as well as other criteria, or with regard to energy use, which we will be using as our $qual$-measure in this paper.

A pump pu injects mechanical energy into a WDN to move water from lower elevations to nodes with higher elevation, which are the aforementioned tanks. A pump, as a directed edge in the graph representing a WDN, creates head at the "receiving" end of the edge dependent on the flow through the pump from the reservoir side. This dependency is described by a so-called *pump curve*, which is also dependent on the energy put into the system by the pump (in case of the commonly used centrifugal pumps this energy depends on the speed of the rotating impeller of the pump [4]). Modern pumps allow for different amounts of energy created by the pump, which means that a pump operator can choose the pump curve he prefers for the current network conditions (achieving the appropriate efficiency of the pump), introducing a second way to influence the energy usage by the WDN (in addition to the choice of pump). Naturally, the mechanical energy put into the WDN by a pump does not equal the electrical energy used by the pump, because of heat loss due to friction.

The influence of a pump and its performance on a particular WDN is governed by a set of complex equations, taking into account the network, the characteristics of all nodes (for example, elevation, demand at junctions, which are normally used to provide exit points for the water), and the characteristics of the edges (for example, pump curves, diameter, or pipe lengths). Furthermore, by opening or closing valves, the network structure can be changed. Fortunately, the operation of a WDN does not really require the continuous solving of this set of equations reacting to changes in demand and trying to find a solution that requires minimal energy (which would require quite some time to compute). A reactive approach looking at the fill levels of the tanks (and using operational minimum h_{min} and maximum h_{max} water levels for each tank) is usually sufficient to meet the demands by the customers. Experienced human operators often also are able to operate the pumps in reaction to the fill levels of the tanks with a "not so bad" energy consumption. Even designing WDNs is nowadays not using the set of equations directly, instead simulators, such as the EPANET toolkit, are used to provide feedback about what is happening in a WDN.

While the main purpose of a WDN is to fulfill the demands by the customers at all times, keeping the energy consumption as low as possible is naturally in the interest of the organization or company responsible for a particular WDN. Since the customer demands are usually not fully predictable, keeping the energy consumption low is a dynamic distributed optimization problem that is well suited for being solved by a distributed

self-organizing system.

C. DIC for WDNs

The use of digital infochemicals allows for fine-tuning the timing of communications and which agents receive information, while naturally having not to know all of the agents within A, hence guaranteeing the flexibility and scalability of the system. [5] made use of this by developing two control modes for WDNs based on DIC, namely the greedy mode (GM) and the coordinated greedy mode (CGM). Since the experiments in [5] clearly showed that CGM is better than GM, we will in the following only present CGM and also extend only this mode by GEIA.

In CGM, the agents in A consist of tank agents $\mathcal{A}g_{tk_i}$ for each tank tk_i in the network, who are constantly monitoring the head h in their tanks, and pump agents $\mathcal{A}g_{pu_j}$ for each pump $pu_j \in Pu$. As already stated, for a tank there are two limits, the minimum head limit h_{min} and the maximum head limit h_{max}. Every $\mathcal{A}g_{tk_i}$ continuously predicts the future head value of its tank based on the past head values and pressure fluctuations using linear regression. This way it can detect that h converges against one of the limits before this limit is actually reached and if this is detected it sends out a so-called *help infochemical* ic_h:

$$ic_h = (\gamma_h, \gamma_h^{thresh}, \delta_h, id_{\mathcal{A}g_{tk}}, q_{tk}),$$

where γ_h, γ_h^{thresh} and δ_h are as before, $id_{\mathcal{A}g_{tk}}$ is the unique id of the emitting tank agent $\mathcal{A}g_{tk}$ and q_{tk} is the current inflow of the tank tk.

A help infochemical, as all infochemicals in the system, is propagated between locations in the environment under the control of a propagation function. This function reduces the concentration γ proportional to the length of the pipe connecting the locations multiplied by the diffusion coefficient δ. As soon as $\gamma < \gamma^{thresh}$, the propagation is stopped. Infochemicals evaporate following a defined evaporation function: in every time step, the concentration γ is reduced depending on the diffusion coefficient δ, again. And, as soon as $\gamma < \gamma^{thresh}$, the infochemical will be removed from the environment.

When a help infochemical ic_h reaches a pump agent $\mathcal{A}g_{pu_j}$ associated with a pump $pu_j \in Pu$, this agent needs to figure out if pu_j is able to help with the request by the tank agent. This means determining the most efficient speed level the pump's impeller would have to be set in order to provide the required help. By taking into account the current head at pu_j's discharge side, the current flow of the pump q_{pu_j}, as well as the information about the inflow/outflow q_{tk_i} included in the perceived help infochemical emitted by Ag_{tk_i} and the given pump curves, $\mathcal{A}g_{pu_j}$ can not only determine if pu_j can be set to a speed level that can produce the additional flow needed by the requesting tank while maintaining the current head at the pump's discharge side, it can also determine the speed level with the best energy efficiency eff_{pu_j} (see [4] for more details about this computation). Before $\mathcal{A}g_{pu_j}$ is changing its impeller settings, it first sends out a so-called *job infochemical* ic_{job}:

$$ic_{job} = (\gamma_{ic_{job}}, \gamma_{ic_{job}}^{thresh}, \delta_{ic_{job}}, id_{\mathcal{A}g_{pu_j}}, id_{\mathcal{A}g_{tk_i}}, eff_{pu_j})$$

where $\gamma_{ic_{job}}$, $\gamma_{ic_{job}}^{thresh}$ and $\delta_{ic_{job}}$ are as before, $id_{\mathcal{A}g_{pu_j}}$ is the id of the pump agent, $id_{\mathcal{A}g_{tk_i}}$ the id of the tank agent that sent out the help infochemical $\mathcal{A}g_{pu_j}$ could react to and eff_{pu_j} is the efficiency that can be achieved by pu_j. This infochemical indicates to all other pump agents that might have gotten the original help infochemical that pu_j is able to fulfill the request at "cost" eff_{pu_j}.

All pump agents wait for a given amount of time to see if they can perceive job infochemicals by other agents. If this is the case, they use a comparison function to determine which agent will in the end react to the initial help infochemical. For CGM, this comparison function, which is the same for every pump agent, simply selects the pump with the best efficiency, if the help infochemical required an increase in flow, and the pump with the least current efficiency if the help infochemical indicated that the tank is in danger to reach its maximal limit. So, the agent that is selected by the comparison function is aware of the fact that it is selected and will make the necessary change to its pump, while all the other agents know that they do not have to do anything. Additionally, the tank agent with the initial help infochemical will also receive the job infochemicals and will therefore know that help is on the way. This tank agent will continue emitting the help infochemical until it has gotten at least one job infochemical that references its request.

D. GEIA for DIC for WDNs

As we will see in this section, instantiating the GEIA concept for a self-organizing system for controlling WDNs based on DIC is mostly straightforward, although the first step of the instantiation is changing the set A of agents in the system to also include agents for junctions. These agents are necessary to provide information about the whole WDN, since tank and pump agents have only a very limited view of the WDN and rather different global situations can look, from the perspective of tanks and pumps, very similar, so that the junction information is needed to distinguish these different situations. As stated above, the quality criterion $qual$ that is of interest in a WDN is the energy consumption by the pumps and the goal is to minimize this consumption. In the following, we will first go through the different steps in the working cycle of the GEIA and then look at the changes to the different types of agents.

The **receive** action of the GEIA is performed by receiving so-called *history infochemicals* from all other agents in regular intervals. These infochemicals follow the standard pattern presented in Section V-A, with ψ being a sequence of history information, with an element for each time step the agent went through since the last time it sent an history infochemical. The concrete information in ψ depends on the type of agent: a junction agent ju sends only the demand $dem_t(ju)$ at the junction for each time step t, a tank agent tk sends its tank level $lev_t(tk)$ and information on when it sent out any

help infochemicals, and a pump agent provides the energy consumed by its pump for each of the time steps (which allows to compute $qual$ of the emergent solution) and additionally the information on all changes of the impeller speed it made.

Given that the GEIA has the information about the histories of all agents for each time step, it is very easy to implement the **transform** action of the GEIA, it essentially just creates a table $GHist$ with the information from each agent for each time step. As we will see in the following, it is not even necessary to partition this table into a sequence of run instances, due to the continuous nature of the work of the WDN.

The **extract** step has to produce a sequence of recurring tasks out of the table $GHist$. As already mentioned, we are interested in sequences of tasks that immediately follow each other, which means that for WDNs a task is the demand to be fulfilled by the network at a particular time step. We will call a sequence of immediately following tasks a pattern, which means that **extract** has to find a set of recurring patterns P^{rec}. While we could allow such a pattern to be any recurring waveform in the demands on a WDN, for our proof-of-concept system we defined a pattern to be a sequence of time steps of constant demands at the junctions. A pattern is recurring, if it or a similar pattern with similar tank levels at the beginning of the pattern occurs at least a given number $minocc$ of times in $GHist$ (respectively the last k time steps of this history).

The similarity measure sim used to compare patterns must reflect that an agent in A has to determine locally that an exception rule should be triggered, respectively stays triggered. Therefore two patterns 1 and 2 are similar if for each junction ju_j at each of the time steps i the relative demand difference between the two patterns (i.e. $|dem_i^1(ju_j) - dem_i^2(ju_j)|/dem_i^1(ju_j)$) is within a demand threshold $thresh_{dem}$ and if for each tank tk_j at each time step i the water level of the second pattern is within a relative interval around the level of the first pattern (i.e. $|lev_i^1(tk_j) - lev_i^2(tk_j)|/lev_i^1(tk_j) \leq thresh_{ta-low}$ if $lev_i^1(tk_j) > lev_i^2(tk_j)$ for a threshold $thresh_{ta-low}$ and $|lev_i^2(tk_j) - lev_i^1(tk_j)|/lev_i^1(tk_j) \leq thresh_{ta-high}$ if $lev_i^1(tk_j) \leq lev_i^2(tk_j)$ for a threshold $thresh_{ta-high}$). Finding the recurring pattern consists then simply of finding all pattern with constant demand in $GHist$ of a given minimal length n_{minpat}, dividing them into sets of similar patterns and checking if the size of the sets is greater than $minocc$.

The step **optimize** requires a method to determine an optimal or at least very good solution to a recurring pattern (as already mentioned, if there are several recurring patterns, each of them is treated as is described in the following). While there are several methods proposed to solve the static control problem of WDNs, nearly all of them come with limitations that we were not willing to accept. Methods that, in theory, are able to generate the optimal solution are usually not able to deal with networks that have more than one pump, which is naturally not acceptable to us (in fact, [9] states that in practice optimal methods can not handle realistic WDNs without simplifications). Nearly all methods for finding approximate solutions have been tried for WDNs, usually using network simulators like EPANET, see [12], to evaluate potential solutions. We followed the approach from [17], which uses particle swarm optimization, modified to use Goal Ordering Structures (see [11]) to better deal with solutions that are not valid. Since we only optimize individual pattern, which realistically are only a few hours long in most WDNs, this optimizer was sufficient for our system. But it was not able at all to produce very good solutions for demand curves that spanned days. The result of **optimize**, i.e. opt^{rec}, is for each pump a sequence of actions with periods of the form $((pump-speed_1, duration_1), ..., (pump-speed_p, duration_p))$ and naturally the energy consumption $qual(opt^{rec})$ that this solution produces. Please note that despite the fact that we use only constant demand pattern, this does not mean that the optimal solution consists of a single period of constant pump impeller speed. Changes to the speed are possible, although we usually see quite long periods of constant speed.

If the energy consumption of the solution for a demand pattern is substantially less than what the self-organizing system A produced in the past, then the step **derive** uses the action sequences for the pumps for this better solution, together with the demands at each time step and the tank levels at each step (both demands and tank levels should not vary much, given that we are only looking at patterns with constant demand) to create a group exception rule gr. For all grs, we have $A_{gr} = A$. If the pattern has a length of $p*$, then the exception rule to gr for a junction agent ju consists only of a sequence of checks that the current demand $dem(ju)$ is similar (using the local part of sim as defined for step **extract**) to the demand at that step in time in the optimal solution. Formally, this means

$sim(dem_1(ju), dem(ju))$;...;$sim(dem_{p*}(ju), dem(ju)) \rightarrow$.

The exception rule created for a tank agent tk is similar to this, namely

$sim(lev_1(tk), lev(tk))$;...;$sim(lev_{p*}(tk), lev(tk)) \rightarrow$,

with $lev(tk)$ being the current tank level. Finally, in contrast, the exception rule to gr for a pump agent is not using any conditions, it is just the direct translation of the action sequence for this agent from opt^{rec} into the rule form from Section IV, formally

$\rightarrow t_0 + 1 : pump - speed_1$, $duration_1 : pump - speed_2$,
$duration_1 + duration_2 : pump - speed_3$,...,
$\sum_{i:=1}^{p-1} duration_i : pump - speed_p$.

Note that p is usually much smaller than $p*$.

send is using an *inform excep rule infochemical* ic_{ier} to inform each of the agents in A about their particular exception rule as part of a group exception rule gr. In addition to the usual elements γ, γ^{thresh}, and δ, its ϵ part gives the id of the GEIA agent and the content ψ consists of the id of the agent the exception rule is for and, naturally, the exception rule as defined above.

All agents in A are using their exception rules as described in Section IV, except that the signals are sent via *cond fulfilled infochemicals* ic_{cf}. This infochemical has as content ψ just an identifier for the particular group exception rule and the

time at which the infochemical was sent (all other parts of the infochemical are as before, which includes ϵ providing the id of the agent sending the signal). Since all agents are involved in every group exception rule, there can only be one rule that is triggered by all these agents, so that cr^i of an agent just selects this rule. Note that no infochemicals are sent out if an agent does not have signals to communicate.

VI. Experimental evaluation

In this section, we evaluate a system based on the instantiation of the GEIA concept for DIC for WDNs described in the last section in several experiments. We first describe the general setup, including the values for all parameters and then present some individual experiments that highlight the potential of the GEIA concept. In the last subsection we report on results for randomly created network demands, which should provide a realistic view of the usefulness of GEIA.

A. Setup

In all of our experiments, a time step is 1 minute. The minimal length for a pattern n_{minpat} was set to 3 hours and we gave the GEIA 48 hours before it became active, with $minocc = 2$. The thresholds for the similarity measure were set to $thresh_{dem} = 95\%$, $thresh_{ta-low} = 8\%$, and $thresh_{ta-high} = 9\%$. The solution opt^{rec} needed to be 10 percent better than the emergent solution $last$ for the GEIA to create a group exception rule. On the infochemicals side, we wanted a quick distribution of the GEIA related messages and therefore selected for all new infochemicals $\gamma = 1.0$, $\gamma^{thresh} = 0.05$, and $\delta = 0.94$.

We used 3 different WDNs in our experiments. The first one, manyP is taken from [5] and was artificially created to have a network with a lot of choice with regard to pumps reacting to tasks from tanks. More precisely, the network consists of 5 pumps placed near 5 reservoirs, 2 tanks, 11 junctions and 15 pipes. All of these components are at elevation level 0. Both tanks have an h_{min} of 9 and an h_{max} of 11, but one tank has a diameter of 30 and the other a diameter of 20. The pumps have different pump and efficiency curves providing, as already stated, a lot of possibilities in the network.

The second WDN is also a small network that was presented in [16] by van Zyl et al. The network has two tanks and two pump stations, one of which has two pumps and the other only one. One tank is located 5 meters higher than the other, which is 70 meters higher than the reservoir. The network also has 6 junctions and 14 pipes of different length and diameter. The tanks have otherwise the same properties and both have an h_{min} of 86 and an h_{max} of 88 (which produces a lot of tasks for the pumps). The two pumps in the first station are identical, while the one in the second station has the same efficiency curve but a different pump curve as the ones in the first station.

The third WDN is the WDN of the city of Novato in California. Figure 2 shows the schematic of this WDN. This WDN comes with the EPANET simulator and supplies water

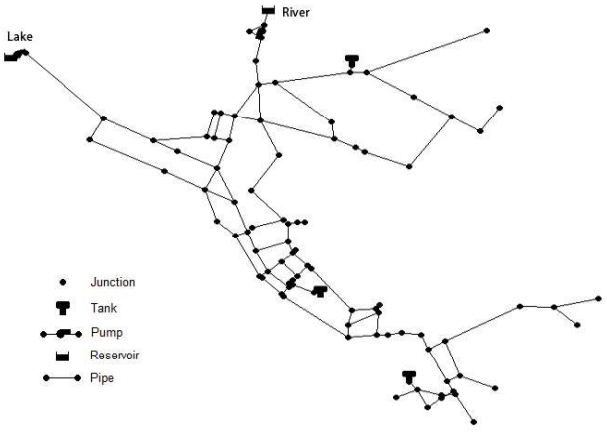

Fig. 2. Novato Municipal Water Distribution Network

Exp.	120h		168h	
	impr. %	kWh	impr. %	kWh
1	16.79%	6.3E5	10.12%	4.80E5
2	12.71%	4.8E5	14.65%	7.60E5
3	13.74%	4.9E5	21.52%	1.12E6

TABLE I
Highlights of comparison without and with GEIA (difference in percent and accumulated energy in kWh) in Novato WDN

for approximately 50,000 people. In the WDN there are 2 pumps, 3 tanks, 2 reservoirs, 92 junctions and 117 pipes. Pump information can be found in [5]. The 3 tanks are different and all have a larger span between h_{min} and h_{max} than in the first two examples. As can be expected for a city, the pipes are rather different in diameter and length.

To give an impression about the optimizer we used, it on average needed 32.6 seconds to create a solution for a pattern for manyP, 88.6 seconds for the Novato network and 354.8 seconds for the vanzyl network. This means that a good solution is easily found within the time of a run instance (which naturally is not a requirement for the approach, but allows for providing fast advice).

B. Selected good examples

In this section, we provide a few examples that show the potential of the GEIA concept. We created these examples out of some good runs from the experiments in the next subsection, but we modified the demands at some junctions for some time steps (usually after agents just had finished following a group exception rule) so that bad transitions back to the reactive behavior were avoided (see the analysis in the next subsection).

As Table I shows, double digit percental improvements are possible, although, as example 1 shows, the improvements are not steady. But example 3 shows that this goes both directions. To get a better idea what the percentages and the kWh really mean, the improvement of 1.12E6 kWh over 168h at a kWh price of $ 0.07399 (which was the household energy price for a kWh in Calgary at the end of November 2012) amounts to

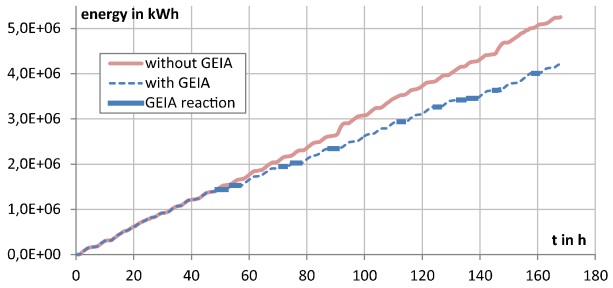

Fig. 3. Accumulated Pump Energies usage for example 3 of Table I

Network	Exp.	120h		168h	
		avg. %	kWh	avg. %	kWh
	1	2.93%	4.90E1	3.73%	1.30E2
manyP	2	4.99%	1.06E2	3.37%	1.04E2
	3	0.67%	1.20E1	1.87%	5.40E1
	1	4.54%	5.10E5	9.74%	1.52E6
Novato	2	8.00%	8.50E5	10.02%	1.50E6
	3	3.60%	3.70E5	8.29%	1.26E6
	1	3.53%	1.60E5	5.15%	3.30E5
vanzyl	2	4.10%	1.90E5	2.81%	1.90E5
	3	4.24%	2.20E5	5.16%	3.90E5

TABLE II
COMPARISON WITHOUT AND WITH GEIA (AVG. PERCENTAGE OF IMPROVEMENT AND ACCUMULATED ENERGY IN KWH)

approximately $ 11,800 per day (which includes the 2 days of "learning" the patterns). The total cost per day for this example for the system without advisor is around $ 55,000.

Figure 3 shows a comparison of the accumulated energy consumption between the basic self-organizing system and the system with GEIA. As can be seen, after the first 48h, in which there is naturally no difference, often when the system with GEIA shows a reaction to a group exception rule, which usually results in very little energy consumption (the bar indicating this usually covers the raise in consumption), we see rather large consumption by the basic system (see, for example, 90h or 145h). Obviously, this is exactly what GEIA is about!

C. Randomly created examples

In order to get an idea about the average behavior of the GEIA, we performed experiments with all three WDNs with (mostly) randomly created demands. They were "mostly" randomly created, since obviously GEIA will not produce any group exception rules, if there are not recurring patterns in the demand. In order to make sure that we had at least one often enough occuring pattern, we created an experiment for a WDN in the following way: we first created randomly a constant demand behavior for the network of length of 3 hours. We then created a random demand behavior of 48 hours around 6 occurrences of this constant demand behavior, respectively similar constant demands with regard to sim (for all junctions). We also created then random continuations of this behavior for an additional 72 hours (with another 6 occurrences of the recurring constant behavior, respectively similar constant behaviors, in it) and for another 48 hours (again, with 6 occurrences of the recurring pattern or similar patterns in it). This provides us with two measuring points, namely after 120 hours (48 hours of "training" followed by 72 hours of application of group exception rules) and after 168 hours (a continuation of an additional 48 hours of applying group exception rules).

For each network we created 3 different initial constant demand patterns and for each pattern we used the above construction of a random demand 3 times to get 3 different runs that form an experiment. Table II reports on the average improvement (in percent) between the system without using GEIA and with using GEIA and we also provide the amount of energy (in kWh) that the system with GEIA saved (accumulated over the 3 runs of an experiment) for the two measuring points. First of all, each of the experiments resulted in an improvement due to GEIA and if we look at the improvement percentages, then the more complex the network the higher is the improvement percentage. While the average improvement percentage from 120h to 168h is not always getting larger (manyP experiment 2 and vanzyl experiment 2 see a decrease), only manyP experiment 2 saw a slight decrease in the saved energy from 120h to 168h, indicating that it is possible that there are periods in the experiments where the system without GEIA can be better than the system using GEIA.

But this has to be expected because of the non-recurring demand tasks that have to be fulfilled after the system has finished executing a group exception rule. If these demand tasks are substantially higher than the demand of the exception rule, this can result in the need to use the pumps at high cost levels to fulfill this demand, while the system without GEIA might have brought the tank levels higher during the rule period than the system without GEIA, so that it then is less costly to react to these high demands. And it is also possible that the demand leading up to the triggering of a group exception rule puts the basic system in a situation where its behavior is not so bad.

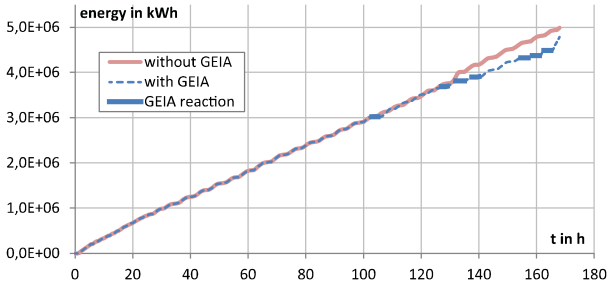

Fig. 4. Accumulated Pump Energies for worst Navato run

Similar to the last subsection, we want to give a monetary interpretation of what the GEIA achieves. If we take the accumulated kWh that the system with GEIA spends less than

the basic system over all the runs for Novato (using the same kWh price) and divide it by the number of days of all these runs (which then includes 9 2-day spans for learning), we get an average of around $ 5,000 spent less per day due to GEIA. While this is naturally substantially less than the maximum gain from the last subsection, it is still quite a lot!

Figure 4 shows the accumulated energy consumption for the worst of the Novato runs we observed. As can be seen, initially the system with GEIA has problems getting group exception rules triggered, resulting in no gains. The first triggering at around hour 102 demonstrates the high energy consumption after the reaction to a rule is over, as we mentioned above (we see a similar behavior at the end), but then in later rule applications we see the largely higher energy use by the basic reactive system.

VII. Related work

While there are several works in the literature that have an agent influence the agents of a distributed self-organizing system, all of these works have this agent taking direct control of the self-organizing system, thus introducing all of the problems associated with a central control (like being a single point of failure). For example, the observer/controller approach of [3] requires the ability to observe and control the underlying system at all times. Management-by-exception, presented in [13] has such a control agent only taking over the system if certain performance conditions are not met. [1] presents a model-based control framework that uses limited lookahead control to optimize the predicted behavior of the underlying system for a limited horizon. The prediction is based on a fixed stochastic model and does not take the actual history of the system into account. In contrast to all these approaches, the advisor never is in control of any of the agents in the self-organizing system it advises, it only provides the agents with an analysis of their past and the ability to change their behavior if it was not efficient.

Due to the highly connectedness of the environment in a WDN, there have been no approaches to use distributed self-organizing systems for their control, with the exception of [5] which we used as basis for our work. In fact, we are not aware of any work that tries to deal with the dynamic nature of this optimization problem. There are quite a number of works, however, that try to solve the static optimization problem, but, as already stated, all of these works put limitations on the networks they were considering. Therefore we were not able to use any of them to create optimal solutions for the examples in the experiments from the last section. Even [16] did not provide such a solution for any demand behaviors for their network.

VIII. Conclusion and future work

We presented an extension of the advisor concept from [14] and [15] to allow to deal with distributed self-organizing systems that act in highly connected environments. Our extension uses pro-active group exception rules that require a group of agents to signal each other their applicability during their use. We instantiated this concept to a system controlling water distribution networks. Our experiments showed both a high improvement potential and solid improvements when faced with randomly created demand behaviors (build around recurring constant demands, which is a requirement for using any kind of advisor).

Since our experiments also showed the potential danger of bad transitions from using rules to using reactive behavior, which was expected given that [6] reported possible exploits of the EIA concept, especially for pro-active advice, future work should look into trying to extend the concept of a risk-aware advisor from [7] to highly connected environments. Naturally, applying the GEIA concept to other such environments, like power grids, is also on our to-do list.

References

[1] S. Abdelwahed and N. Kandasamy, "A control-based approach to autonomic performance management in computing systems", in S. Hariri, M. Parashar (eds.), Autonomic Computing, CRC Press, 2006, pp. 149–167.

[2] S. Brückner, Return from the Ant - Synthetic Ecosystems for Manufacturing Control, PhD thesis, Humboldt-Universität Berlin, 2000.

[3] 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. 2nd ISOLA*, IEEE, 2006, pp. 185–191.

[4] P. Cooper and G. Tchobanoglous, "Performance of Centrifugal Pumps", in *Pumpin Station Design*, 3rd ed., Elsevier, 2008.

[5] F. Dötsch, J. Denzinger, H. Kasinger, and B. Bauer, "Decentralized real-time control of water distribution networks using self-organizing multi-agent systems", in *Proc. SASO'10*, Budapest, 2010, pp. 223–232.

[6] J. Hudson, J. Denzinger, H. Kasinger, and B. Bauer, "Efficiency Testing of Self-adapting Systems by Learning of Event Sequences", in *Proc. ADAPTIVE'10*, Lisbon, 2010, pp. 200–205.

[7] J. Hudson, J. Denzinger, H. Kasinger, and B. Bauer, "Dependable Risk-Aware Efficiency Improvement for Self-Organizing Emergent Systems", in *Proc. SASO 2011*, Ann Arbor, 2011, pp. 11–20.

[8] H. Kasinger, B. Bauer, and J. Denzinger, "Design Pattern for Self-Organizing Emergent Systems Based on Digital Infochemicals", in *Proc. EASe'09*, San Francisco, 2009, pp. 45–55.

[9] M. Lopez-Ibanez, T. Devi Prasad, and B. Paechter, "Ant colony optimization for optimal control of pumps in water distribution networks", Journal of Water Resources Planning and Management, 134(4):337–346, 2008.

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

[11] T. E. Mora, A. B. Sesay, J. Denzinger, H. Golshan, G Poissant, and C. Konecnik, "Fuel optimization using biologically-inspired computational models", in *Proc 7th International Pipeline Conference*, Calgary, 2008, pp. 167–173.

[12] L. A. Rossman, "Epanet users manual", EPA-Environmental Protection Agency, 1994.

[13] 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.

[14] J.-P. Steghöfer, J. Denzinger, H. Kasinger, and B. Bauer, "Improving the Efficiency of Self-Organizing Emergent Systems by an Advisor", in *Proc. EASe'10*, Oxford, 2010, pp. 63–72.

[15] T. Steiner, J. Denzinger, H. Kasinger, and B. Bauer, "Pro-active Advice to Improve the Efficiency of Self-Organizing Emergent System", in *Proc. EASe'11*, Las Vegas, 2011, pp. 97–106.

[16] J. E. van Zyl, D. A. Savic, and G. A. Walters, "Operational optimization of water distribution systems using a hybrid genetic algorithm", Journal of Water Resources Planning and Management, 130(2):160–170, 2004.

[17] C. Wegley, M. Eusu, and K. Lansey, "Determining pump operations using particle swarm optimization", In *Water Resources 2000*, ASCE, 2000, pp. 206–212.