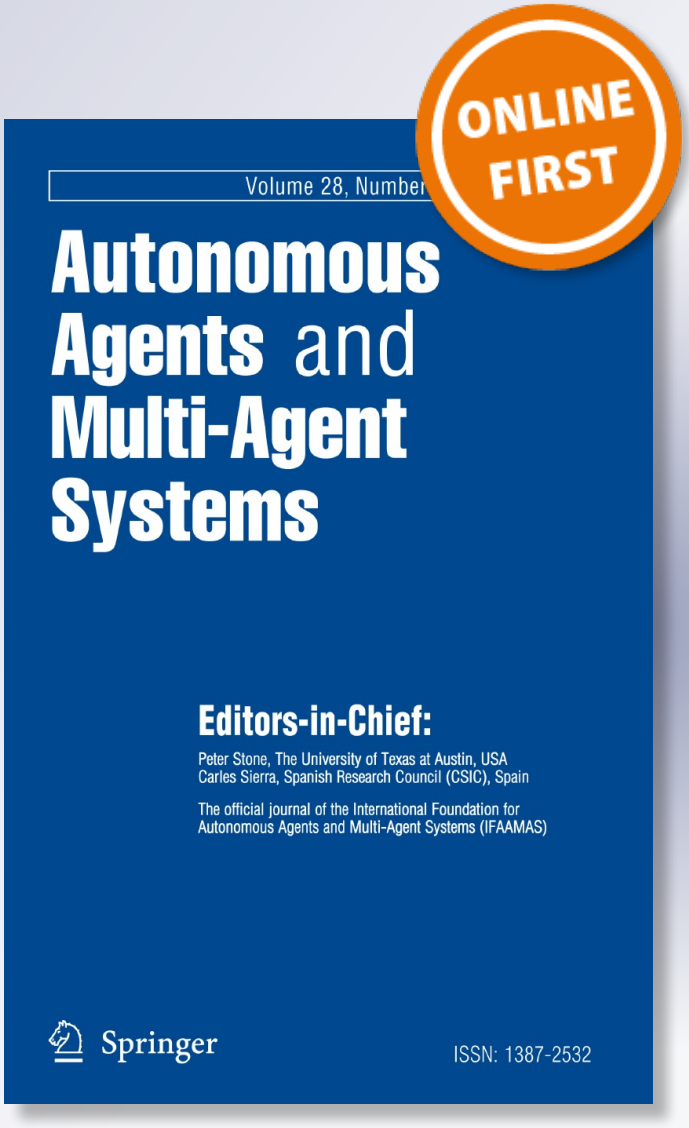
Risk management for self-adapting self-organizing emergent multi-agent systems performing dynamic task fulfillment

Jonathan Hudson & Jörg Denzinger

Autonomous Agents and Multi-Agent Systems

ISSN 1387-2532

Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-014-9274-0



Springer

Your article is protected by copyright and all rights are held exclusively by The Author(s). This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

Author's personal copy

Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-014-9274-0

Risk management for self-adapting self-organizing emergent multi-agent systems performing dynamic task fulfillment

Jonathan Hudson · Jörg Denzinger

© The Author(s) 2014

Abstract The goal of self-adapting self-organizing emergent multi-agent systems applied to problems with dynamically appearing tasks is to reduce operation and design costs. This is accomplished through the design of autonomous agents, which interact to produce behaviors required for flexible and scalable operation. However, when combined with agent autonomy, emergent behaviors are unpredictable resulting in a lack of trust for applications desiring efficiency such as logistics. An additional consultation agent, known as an efficiency improvement advisor (EIA), has been shown to increase efficiency through autonomy preserving advice provided as exception rule adaptations to agents. The problem addressed in this paper is that, in order for EIA-adapted systems to be deployed, the stakeholders must be assured that the risks of both autonomous and adapted behavior are properly assessed and managed. This paper presents a complete framework for a risk-aware EIA (RA-EIA) which uses reflection in order to manage the risks associated with autonomous agents and prospective adaptations. Monte Carlo simulation is used to reduce the frequency of emergent misbehavior appearing during regular operation. Meanwhile, an exploratory testing method, termed evolutionary learning of event sequences, is used to deal with the possibility of severe emergent misbehavior as the result of an malicious adversary or a series of unfortunate events. The experimental evaluations and accompanying descriptive example, for the application area of logistics via pickup and delivery problems, demonstrate that the risk-aware adaptations provided from consultation with the RA-EIA agent allow the client system to be trusted for long-term independent and reliable operational efficiency.

Keywords Risk assessment · Risk management · Evolutionary testing · Emergence · Reflection · Efficiency · Multi-agent systems

J. Hudson (✉) · J. Denzinger
Department of Computer Science, University of Calgary, Calgary, Canada
e-mail: jwhudson@ucalgary.ca

J. Denzinger
e-mail: denzinge@cpsc.ucalgary.ca

Published online: 14 September 2014

Author's personal copy

1 Introduction

Self-organizing multi-agent systems provide a flexible solution to many interesting distributed system problems with an unpredictable dynamic nature [14]. The design goal of these systems is for the collective behavior emerging from the autonomous components' local interactions to provide a solution to the global problem [31]. When problems are both spatially time sensitive and dynamic in nature then distributed multi-agent solutions consisting of autonomous agents are able to respond locally to newly revealed information in a timely manner. There are a number of benefits to employing a self-organizing system over a central solution beyond that of flexibility. First, each autonomous agent is often less complex and cheaper to design, manufacture, and/or operate which increases scalability. Second, the local problem solving of individual agents is computationally and/or operationally more efficient by avoiding the communication/waiting costs required to inform and retrieve a response from a central control. These benefits are often positive financially, resulting in fewer costs associated with system design and operation.

However, in many real-world applications strict self-organizing systems are not a viable option due to the risks associated with their implementation. These risks are associated with the trade-off that, since the autonomous agents of a self-organizing system are unaware of the complete global problem, they are inherently incapable of ensuring global efficiency. In contrast to a pure central solution, which admittedly may not be a possible solution to many problems, self-organizing agents are only able to optimize the local sub-problems they are aware of. Additionally, the behavior that emerges from the collective self-organizing system is unpredictable due to the self-directed nature of the individual agents. This unpredictability is exacerbated if the system is capable of self-adaptation, an optional property that allows the individual agents to change to better suit an identified problem. The result is that the financial benefits gained from design are offset by the unpredictability and lack of controllability of the self-organizing system making it an unappealing option for cost-aware applications.

This paper attempts to address the viability of self-organizing emergent systems for cost-aware applications where there are risks associated with the unpredictable emergent behavior of autonomous agents. The risks of emergent misbehavior are elusive, as even a small change in an agent can lead to unpredictable and drastic changes in the behavior of the complete system [36]. This is only increased by the addition of self-adaptation. Self-adaptation can increase performance efficiency. However, in order to guarantee reliability it is necessary to address the corresponding risk of it creating additional emergent misbehavior. The contribution of this paper is the introduction of a complete framework for a consultation agent, called the Risk-Aware Efficiency Improvement Advisor (RA-EIA), which is capable of assessing and managing the risks of both regular operational misbehavior and severe exploitation misbehavior. A descriptive example which motivates the use of the RA-EIA and demonstrates its operation for the application area of PDP is described in Sect. 8.2.

An initial and incomplete concept of the RA-EIA was introduced in [19] and was designed around assessing and managing only the risk of regular operational misbehavior through Monte Carlo Simulation (MCS). In this paper, the explanation of this initial framework has been expanded and, more importantly, additional evaluations exploring its capabilities have been completed. However, this initial concept did not address the severity of risk from exploitation, which is necessary to manage the complete range of types of risk. The more complete RA-EIA framework described in this paper addresses this by assessing and managing the severity of the extent of possible emergent misbehavior by integrating and automating the Evolutionary Learning of Event Sequences (ELES) method from [18] in a resource-aware manner. The combination of these two assessment methods is used to quantitatively deter-

Author's personal copy

mine a set of risk-averse adaptations designed to improve the efficiency of the emergent multi-agent system consulting the RA-EIA. The previously mentioned example is expanded in Sect. 8.4.1 to demonstrate this risk-averse adaptation performed by the RA-EIA for the application area of PDP.

Multi-agent systems are an obvious solution for creating self-organizing systems for problems with dynamic and unpredictable natures. More particularly, they are a solution for problems consisting of completing a set of tasks that can appear unpredictably during system operation. These problems will be termed Dynamic Task Fulfillment (DTF) problems [6]. One solution to solving Dynamic Task Fulfillment problems is found in the Pollination Inspired Coordination (PIC) self-organizing coordination method used in the evaluations of this paper. PIC is an example of using a natural paradigm to inspire a self-organizing multi-agent system design pattern [25]. The information technology analogues of such natural paradigms generally have important differences. For example, in a biological system movement costs may be relatively cheap and this may not hold for the industrial counterpart.

Importantly, in most industrial applications of task fulfillment problems it is desirable to assure the operator of a complex system of certain levels of global efficiency. For example, in many logistical problems minimizing travel costs and reduplication of effort is beneficial. The difficulty of promoting performance efficiency in self-organizing emergent multi-agent systems has been explored under the topic of self-adaptation. Self-adaptive systems adjust their behavior for better performance [57]. The addition of a special Efficiency Improvement Advisor (EIA) consultation agent to a client emergent multi-agent system solving task fulfillment problems has been shown to assist the self-adaptation of the other agents [49,51]. When non-intrusive communication is possible, this EIA agent provides exception rule advice to improve the collective efficiency of the system based on the assumption that the problem has recurring patterns the EIA agent can discover and derive advice from. This paper addresses the problem that, for cost-aware applications, there is often an unacceptable level of inefficiency risk resulting from the combination of the EIA agent's adaptations with the emergent behavior of an autonomous multi-agent system.

Outside of emergent misbehavior that may result during regular operational circumstances, it is important to consider the danger of malicious adversaries, or simply a series of unfortunate events, which exploit the advised client system to create inefficiency. The considered malicious behavior consists of manipulating the problem the agents encounter, not directly manipulating the agents themselves. Exploitation is a unique type of risk that must be assessed and managed for certain application areas of task fulfillment, either due to the possible operational costs associated with it, or the requirement of liability management. This risk of exploitation may be driven by parties such as casual trouble-makers, aggressive criminal organizations, rival companies, or unhappy employees. The paper [18] introduced a human-managed evolutionary testing method, termed Evolutionary Learning of Event Sequences (ELES), designed to expose misbehavior in emergent systems.

In one practical example of a malicious adversary a delivery logistics company is expected to go up for sale in the near future. This logistics company uses a self-organizing multi-agent system design to deliver packages and an EIA agent to increase the efficiency of its operations. However, a rival company has knowledge that this system is not capable of assessing and managing the risks associated with its operation. Using this knowledge a rival is capable of exposing the deficiencies of the delivery system (eg. by introducing specifically timed delivery requests) which exploit the adaptations provided by the EIA to specific delivery agents. The demonstrated existence of these deficiencies, or simply the consequences of the resultant unacceptable delivery performances, will trigger a corresponding drop in valuation and thus share price of the company up for sale. Consequently, a considerable savings during

Author's personal copy

the subsequent auction process is possible therefore justifying the time, effort, and cost investment of determining how to accomplish the exploitation.

The RA-EIA's risk management provides assurance to the operators that the employed system's efficiency can be trusted to perform within a chosen reference of performance. These claims are demonstrated by instantiating the RA-EIA for the efficiency-aware logistical problem of dynamic PDP, see [21], which consists of the service of delivery demands by a limited set of transportation agents [54]. Self-organizing emergent multi-agent systems are well-suited to the dynamic nature of such on-demand logistical problems. Nonetheless, the risk of uncertainty and the inability to guarantee reliability has prevented these systems from being applied in the real world. Providing assurances against risk in order to remove this uncertainty and increase reliability is the ultimate goal of the research presented in the paper.

The remainder of the paper is structured as follows. First, Sect. 2 defines self-organizing emergent multi-agent systems and the DTF problems they are used to solve. Next, in Sect. 3 related research areas will be covered. Section 4 introduces the EIA, the self-adaptation component of the system. The process of ELES is described in Sect. 5. An abstract definition of the RA-EIA component is described in Sect. 6. This is followed, in Sect. 7, with an instantiation for PDP solved using PIC. In Sect. 8, the results of evaluating the instantiated system are discussed. These results include an examination ability to assess both the expected frequency and the extent of the severity of risk. Accompanying the experimental evaluations is a progressive example which, in Sect. 8.2, demonstrates the use of the RA-EIA to increase efficiency by advising exception rule adaptations to agents. This example is expanded in Sect. 8.4.1 to explore the RA-EIA's capability to manage the risk of these exception rules being exploited by an adversary. Lastly, Sect. 9 follows with conclusions and future work.

2 Background

This section defines self-managing multi-agent systems and the general class of Dynamic Task Fulfillment (DTF) problems they can be utilized to solve. The purpose of self-management is to free system operators from the minor details of system operation [31]. Such self-directing systems are often termed autonomous systems and expected to act without human direction [55]. Self-management is also sometimes referred to as autonomicity which requires the system to be responsible for achieving certain specified goals such as efficiency [53]. Generally there are two approaches to achieving self-management: bottom-up approaches composed of a large number of self-organizing elements which autonomously change based on requirements, and top-down approaches utilizing a control loop that enables self-adaptation based on specified high-level objectives [23,58].

Self-adapting self-organizing emergent multi-agent systems are well-suited for many self-managing system applications. A multi-agent system MAS can be defined as a pair (A, Env) consisting of a set of agents $A = \{Ag_1, \ldots, Ag_m\}$ and a set describing the possible environment states Env. A multi-agent system's communication and coordination mechanisms are embedded within the agents' definitions. An agent Ag can be represented by a quadruple (Sit, Act, Dat, f_{Ag}) where Sit is a set of situations an agent can be in, Act is the set of actions an agent can perform, Dat is the set of possible value combinations that an agent's internal data areas can have, and $f_{Ag} : Sit \times Dat \rightarrow Act$ is the decision function an agent uses in order to determine its next action.

Self-organization enables distributed systems to be self-managing [7]. Self-organizing systems are designed around exploiting emergence [47]. Emergence is the result of interacting elements of a system demonstrating new characteristics in combination other than those

Author's personal copy

found in an individual element. On the other hand, self-adaptation in a system is generally accomplished through two methods: parameter adaptation of program variables and compositional adaptation by adding new elements/algorithms. Control theory can be considered an area of self-adaptation that deals with managing the output of a dynamic system via parameter adaptation to achieve a reference of performance [40].

Hybrid self-adapting self-organizing emergent multi-agent systems bring together the strengths of both methods. However, since self-organizing emergent behavior cannot be strictly determined from examining an individual agent, it is desirable to have some method of predicting system wide consequences resulting from the adaptation of single agents. The risk assessment and management used by the RA-EIA, described abstractly in Sect. 6 and instantiated in Sect. 7, utilizes predictive performance evaluation to assess the risks associated with adaptations to a self-organizing emergent multi-agent system.

Assessing risk begins with determining a valuation of some quality or quantity associated with a system. Cost is associated with a differential change in this value as the result of a decision such as an adaptation. Loss is therefore a negative differential and gain a positive differential. There are two general approaches taken to assessing the risk of loss:

1. Examine the statistical distribution associated with multiple gain/loss outcomes to determine the expected loss given a change to the system.
2. Examine the extent of loss that would occur if a particular costly outcome of an event occurs. Risk is considered according to the probability and related cost of the event.

The assessment of risk is dependent on the type of problem being considered. Self-managing systems are well-suited for dynamic distributed problems where traditional central coordination is undesirable. DTF problems are one where the system begins to solve revealed portions of a problem before the remaining parts are known. Many self-managing systems will solve multiple problem instances over a period of time. These multiple problem instances can be represented by a run sequence.

A run sequence rs of k run instances can be defined as $rs = (ri_1, ri_2, \ldots, ri_k)$. A run instance ri is a sequence of dynamic events $(ev_1, ev_2, \ldots, ev_m)$ the system encounters during an interval $Time$ of execution. Finally, an event ev is a task-time pair $(ta, t_{\text{ta}}^{\text{avail}})$, where $t_{\text{ta}}^{\text{avail}} \in Time$ is the point at which task ta becomes available to be solved. A task consists of a pair $(Prop, [t^{\text{start}}, t^{\text{end}}])$ of the properties $Prop$ and time interval $[t^{\text{start}}, t^{\text{end}}]$ for completion of a task.

A solution for a run instance consists of assignments of events to agents. An assignment of an event ev_i to an agent Ag_j is defined as a triple (ev_i, Ag_j, t_l), where the event ev_i will be started by $Ag_j \in A$ at the point in time $t_l \in Time$. The set of all possible assignments is denoted as $Assign$. Note that completing an event ev_i may require a sequence of actions by an agent Ag_j. Depending on the domain and system, agents may not be able to perform certain tasks.

Lastly, a solution sol generated by a set of agents A for a particular run instance ri that consists of several events $(ev_1, ev_2, ..., ev_m)$ is defined as $(as_1, as_2, \ldots, as_m)$, where each assignment $as_i \in Assign$ corresponds to an event $ev_i \in ri$. The set of all possible solutions is denoted as Sol. The system's actual solution to a particular run instance is known as the emergent solution. A solution is generally considered to be of a certain quality $qual$ (i.e. cost, time), which is dependent on the particular problem, application domain, and system. For example, a solution quality metric may be the total distance traveled by the agents to complete a run instance.

The RA-EIA is evaluated for the application area of PDP, a subclass of DTF. The example in the experimental evaluations, in Sect. 8.2, demonstrates a run instance formed of package

Author's personal copy

delivery events in a grid environment. Additionally, an emergent solution to this run instance, along with an estimated optimal solution, can be found in this section.

3 Related work

This section describes related research areas beginning with self-adaptation via control theory. Feedback control loop concepts have been advantageously adopted from control theory by computer science. Control loops typically involve the key activities: collect, analyze, decide, and act [9]. Examples include Shaw's self-management via self-adaptation [48], Organic Computing's observer/controller architecture [42], management-by-exception [46], and IBM's MAPE-K autonomic manager [20]. Unlike the EIA and RA-EIA approach described in this paper, these systems represent the use of a central control and therefore infringe on the flexibility and autonomy of the components they manipulate.

In order to automate risk assessment a system must be capable of examining its own operation. One form of self examination, called computational reflection, is used to form reflective systems and to assist learning [33]. Examples include meta-reasoning for an agent to reason about other agents and itself [41], the reflective multi-agent system AALAADIN [11], and REFCON for implementing Agent Communication Contexts [8]. Reflection in self-adaptation for autonomic computing is discussed in [15]. The RA-EIA represents a unique instantiation of reflection for the risk-aware consideration of prospective exception rules to advise.

Evaluating the performance efficiency of self-adapting and/or self-organizing emergent systems through testing is a developing area. A method for propagating changes to an autonomic system's run-time test model when structural adaptations are made is described in [1]. This requires a specifically tailored regression test model, and is designed to find errors/bugs introduced by modifications. Self-testing is another area attempting to enable a system to test itself [32]. However, it again relies on regression tests developed alongside the tested system during design. These testing methods focus on a pass/fail methodology unlike the RA-EIA which considers a stochastic distribution of possible efficiency due to behavior.

Unlike simple control theory which manages cause/effect behavior of variables in essentially real-time, the RA-EIA must be capable of managing the cause/effect risk of compositional adaptations while having infrequent opportunities for interaction. MCS is a demonstrated method of assessing stochastic distributions of behavior as the result of compositional adaptations. For example, MCS has been used in examining the risks of software schedules [60], petroleum well drilling [4], supply chain management [45], transport infrastructure [43], alternative locations of airport construction [44], and project schedules [35]. In regards to autonomic systems, in [37] dynamic server allocation using MCS has been examined.

In regard to the PDP implementation of the RA-EIA for evaluation, many solutions have been developed for PDP. Most are found in Operations Research (OR), but many are also from multi-agent systems, particularly Distributed Artificial Intelligence. Centralized solution methods for static PDP are not flexible enough for dynamic situations [3]. Complete decentralization through cooperating agents provides advantages including scalability and flexibility. For example, the field-based approach found in [56] is a multi-agent solution. Decentralization also has disadvantages; the primary ones are the lack of an optimal solution and limited controllability. Lastly, decentralized systems sometimes exhibit unforeseen and undesirable unwanted emergent behavior.

Hybrids of central control and decentralized agents, such as using the EIA/RA-EIA to advise a decentralized multi-agent system, are the preferred compromise. Hybrid methods

Author's personal copy

include Distributed OR methods, centralized multi-agent system optimization [10], a priori optimization with operational re-planning [5], and embedded optimization [2]. These approaches have shown theoretically or experimentally to outperform strictly centralized or decentralized approaches with dynamic PDP [34].

Finally, in regards to the ELES testing method, in exploratory testing software testers are generally in charge of learning during the process of their testing [24]. Exploratory testing is both random and heuristic. The tester can either explore all areas of the tested system randomly to determine unexpected problems, or rely on experience to target certain areas more productively. To address this time-consuming and repetitive process knowledge-based search can transition the software tester's role away from menial tasks to one of properly managing intelligent testing tools. The ELES method described for use by the RA-EIA and the experimental evaluation is a form of automated evolutionary exploratory testing. A process of using evolutionary algorithms for testing, but only for single autonomous agents, and not a complete emergent self-organizing system, is described in [38]. Another method for exploratory testing, this one via particle swarm optimization, is described in [12].

4 Efficiency improvement advisor (EIA)

This section describes the previously developed Efficiency Improvement Advisor (EIA) which, in Sect. 6, will be improved upon by the introduction of reflective risk assessment and management to create the RA-EIA. Self-organizing emergent multi-agent systems are unable to guarantee run-time efficiency due to lacking global knowledge, their individual agents' reactivity, and the unpredictability of dynamic problems. Standard closed feedback loops found in control theory cannot fulfill the desire for autonomy and handle the difficulties of observability and controllability. However, many applications for self-organizing emergent systems call for a certain degree of efficiency. Therefore, some method of autonomicity (self-management of performance) is desired. The EIA, introduced and explored in [25,28,49,50], is a hybrid self-management solution that combines the beneficial properties of a self-organizing distributed system with a self-adaptive mechanism. A description of the EIA model follows in Sect. 4.1, while the actions of this model are elaborated on further in Sect. 4.2. Finally, Sect. 4.3 introduces the exception rules used by the EIA to adapt the client system. An example showing the application of the EIA to a run instance problem with recurring events for the area of PDP is described in Sect. 8.2.

4.1 EIA model

The EIA is an additional consultation agent added to the existing set A of agents forming the client multi-agent system. The EIA provides the capability of self-adaptation to a self-organizing emergent multi-agent system designed for DTF problems. This self-adaptation is accomplished through exception rule advice communicated to individual agents. The EIA agent improves the quality of the system's solution over a run sequence given the existence of a set of recurring events between individual run instances. Most importantly, the EIA maintains the desirable properties of responsiveness and flexibility found in the base multi-agent system, while improving efficiency through EIA-advised self-adaptation. Note, the base system is never dependent on the EIA agent during online operation, as interaction between the EIA and the other agents occurs offline from system execution.

The utilization of the EIA concept necessitates the meeting of the following requirements:

Author's personal copy

1. The agents in A require both the capability and opportunity to transmit their local histories to the EIA.
2. The agents in A are modifiable by exception rules.
3. The problem instances to be solved must contain a set of recurring events, allowing the EIA a limited opportunity to predict the future.
4. Since the derivation of exception rule advice takes place offline, possibly over several run instances, the set of recurring events must remain stable long enough that the exception rule advice remains valid upon communication.

The first two requirements account for the difficulty in observing and controlling the agents of a self-organizing emergent multi-agent system.

Central control, such as that found in traditional control theory, interferes with the autonomy, flexibility, and responsiveness of the controlled distributed system. Therefore, the consultant EIA agent has been developed to be non-invasive to the online operation of the base client system. The EIA examines the information collected from the agents. If some agents are slower than others to transmit this information, then the EIA waits for this information. This functionality is like a consultant providing efficiency advice to a company. The consultant asks for and then collects performance statistics and information, then as the company continues to operate the consultant examines the information to determine changes which are then communicated back to the company. The provided local history should allow the EIA to determine the environment of the agents. This environmental data is necessary for the EIA to determine what other more efficient options were available for agents. Sufficient facsimile of the real world is always a challenge for simulation in computer science but one that has been accomplished for a variety of problems including the examined PDP application area [54].

The advised exception rules enable a limited ability to guide agent behavior during future online operation. The agent's decision function for determining actions is modified by exception rules. In the general case, advised rules represent knowledge designed to improve the global efficiency of the system in solving repeated problems with repetitive events. Therefore, the agents which are only capable of knowing the local problem should not reject advised rules that are designed for the global problem. However, this depends on the application. It is possible that the more of the global problem the agent is aware of for a specific application, the greater an agent's ability to determine the applicability of rules independently. Importantly, agents remain self-deterministic and able to ignore exception rules that don't apply during operation to a particular run instance.

The last two requirements are necessary for the EIA concept to be applicable to a dynamic problem. These requirements can be considered as representative of a class of self-organizing problems in which a self-organizing system would benefit from the system designer or a consultant examining the system to make changes to increase efficiency for problems with recurring patterns which are inefficiently solved by the default configuration. The EIA takes the place of this service and is especially suited when the recurring pattern changes over time and when the modification process is used multiple times.

To overcome the inability to concretely predict the future it is necessary to examine the past for patterns (i.e. recurring events). Recurring events represent events that were determined to be stable and have repeatedly occurred in the history of past events. If it can be determined that the response to this past set of recurring events was inefficient, then the EIA agent can attempt to derive exception rules for the agents designed to improve the efficiency of the system in the future. The set of recurring events is not required to be static, but must remain stable enough that the EIA's advice is still relevant when it is received.

Author's personal copy

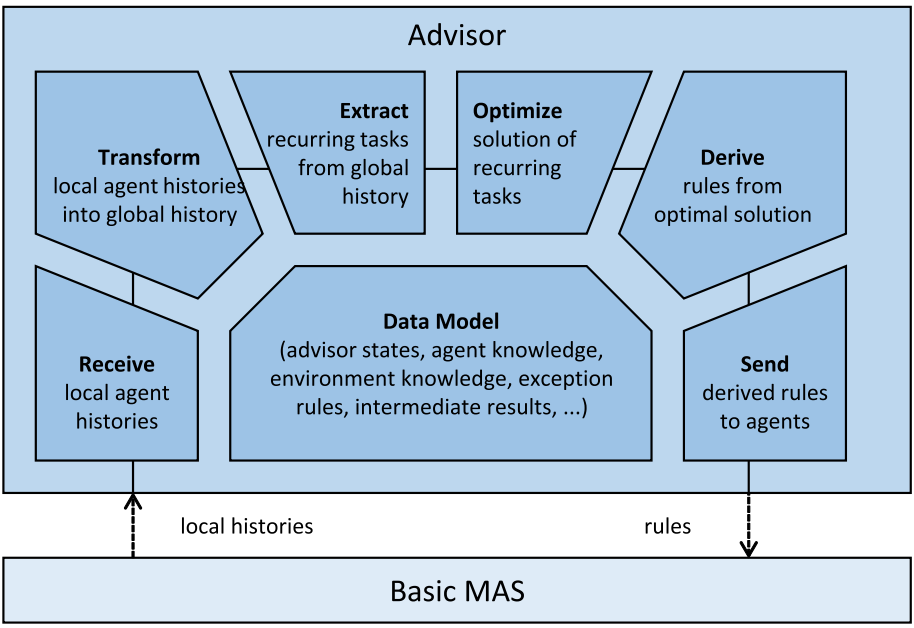


Fig. 1 Functional architecture of EIA

One characterization of multi-agent system architectures divides agents into: deliberative agents that create a plan of action via reasoning and planning based on recorded knowledge of the environment and themselves, and reactive agents who utilize stimulus/response by monitoring the environment and choosing an suitable action [59]. The EIA agent is a deliberative agent that upon consultation advises the reactive agents in the base self-organizing emergent multi-agent system. The resultant heterogeneous multi-agent system is an advised multi-agent system.

The EIA is able to improve efficiency without infringing on agent autonomy by implementing the four activities of a closed feedback control loop, but defined as six distinct functions (see Fig. 1).

4.2 EIA actions

The communication/interaction between an agent Ag_i of the client multi-agent system and the consulted EIA agent Ag^{EIA} is depicted in Fig. 2. The actions **receive, transform, extract, optimize, derive**, and **send** described in this section are all part of a super-action **advise**.

4.2.1 Receive action

The action **receive**(Ag_i, H_i), collects the local history H_i of each agent Ag_i when it is non-disruptive to communicate and stores them in the internal data structure of the EIA.

This local history H_i consists of two types of data:

– Environment data: The local history H_i of an agent $Ag_i \in A$ includes information about experienced situations $sit \in Sit_i$. Given adequate perceptive ability by the agent, the EIA

Author's personal copy

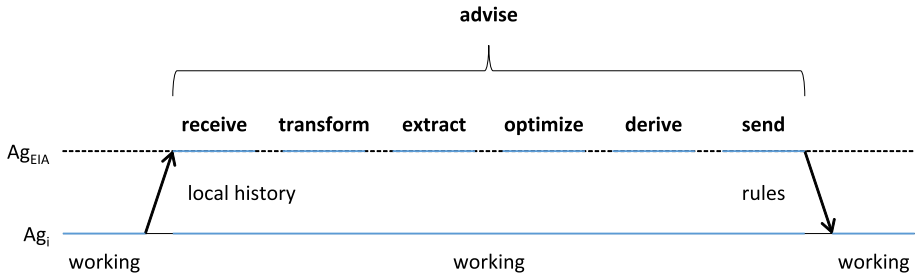


Fig. 2 Interaction between agent and EIA

should be able to recreate, if necessary, the whole environment Env of the multi-agent system A.

– Agent data: The local history H_i includes information on all actions $act \in Act_i$ the agent performed, as well as its corresponding data states $dat \in Dat_i$ at the times of these actions.

4.2.2 Transform action

The action **transform**$(H_1, \ldots, H_n) = GHist$, deduces the global history $GHist$ of the system from the received local histories H_i. The goal is to provide the EIA with a global view on the past run instances. A full reproduction of the ordered system events is not required, but rather certain details about the tasks that had to be solved during the run instances and the assignments of these tasks to the agents of the basic system is required. The starting point for the EIA is the deduction of the events encountered by each agent from its perceptions and resultant actions included in the local history of the agent. The overall global history is a combination of these events.

The global history $GHist$ is a run sequence of run instances $(ri_1, ri_2, \ldots, ri_k)$ that A has solved so far and A's emergent solution sol_{emg} for each run instance. The emergent solution is a set of assignments which indicate which agent responded to which event at what time. The global history is the only reliable source of global context that the EIA has on which to base rule decisions, given the limits of observability. Unobserved, or otherwise undocumented events, will not be captured in the agents' local histories and furthermore not appear in the global history produced by **transform**.

4.2.3 Extract action

The extract action is defined as **extract**$(GHist) = ri^{rec}, NR$. The extract action identifies the ordered set $ri^{rec} = (ev_1^{rec}, ev_2^{rec}, \ldots, ev_m^{rec})$ of recurring events in the $GHist$ as well as the disjoint set NR containing the remaining events not considered recurring $(ev_i^{rec} \notin NR)$. Data from a period of recent run instances in $GHist$ is used to determine events that reoccur. All recurring and non-recurring events can be found in the run instances of the global history $GHist$. If an event from the relevant period of the global history $GHist$ is not in the set ri^{rec}, then it is in the set NR of non-recurring events.

The **extract** action may be accomplished through a variety of means. Note, "*recurring*" may be a fuzzy definition allowing a variation of an event between run instances of a sequence. This is possible through an application dependent similarity measure sim and having a threshold value $minocc$ for how often the abstract event has to occur to be recurring. Using

Author's personal copy

a limited period k, for which to observe the past, allows for the recurring events to change. Data mining the last k run instances with a clustering algorithm such as Sequential Leader Clustering [13] is one method to accomplish the action. If the set of recurring events changes faster than k, then the EIA agent will not be able to detect recurring events and the multi-agent system A will operate independently of the EIA.

The emergent solution sol^{rec}_{emg} to the set of recurring events is a set of assignments. These assignments are a subset of the assignments from the emergent solution to the related run instance. Only the assignments which relate to the recurring events are included. The cost of this emergent solution is based on the evaluated cost required for the agents to perform the indicated assignments. If assignment costs of the recurring events are independent from the non-recurring events which are not included, then the costs the agents reported can be totaled. However, if the assignment costs are dependent, such as common with the Package Delivery Problem, then an estimation can be performed to calculate the distance necessary for the agents to perform the assignments.

4.2.4 Optimize action

The primary objective of the action $\textbf{optimize}(ri^{\text{rec}}) = sol^{\text{rec}}_{\text{opt}}$, is to calculate an estimated optimal solution $sol^{\text{rec}}_{\text{opt}}$ for the set of recurring events extracted by the previous action. This reduces the original problem from a dynamic problem to a static one. However, the determination of an optimum for even the static variant of many computationally hard (i.e. interesting) problems is infeasible. For example, for many problems that invite the application of self-organizing emergent systems, such as the Package Delivery Problem, this is an NP-hard problem. Therefore, algorithms to approximate the solution, such as genetic algorithms, simulated annealing, or tabu search are required.

This estimated optimal solution to the set of recurring events is compared with A's emergent solution $sol^{\text{rec}}_{\text{emg}}$ to the set of recurring events. If this comparison is less than a chosen threshold $\frac{qual(sol^{\text{rec}}_{\text{emg}})}{qual(sol^{\text{rec}}_{\text{opt}})} < qual_{\text{threshold}}$, then no rules need to be derived and the work of the EIA is complete. This threshold is application dependent and generally derived based on performing a cost benefit examination for the chosen solution quality metric. Particularly, it is related to two things. First, the point at which the cost or effort required to improve the system's performance further is not worth it. Second, the amount of variation in performance due to the independence of the agents. This variation generally can not be improved without restricting the agents' autonomy unacceptably.

4.2.5 Derive action

The action $\textbf{derive}(sol^{\text{rec}}_{\text{opt}}, sol^{\text{rec}}_{\text{emg}}, ri^{\text{rec}}, NR) = rules$, derives for each agent Ag_i a set R_i of exception rules from the set ri^{rec} of recurring events such that $R_i \subseteq rules$. The EIA contains a set of rule derivators, each associated with an exception rule variant. The EIA's rule derivators are ordered by priority. The priority of the rule derivators is the decision of the system designer. When deriving advice for a system which we want to retain flexibility the priority is based on using derivators that produce less restrictive rules first.

The **derive** action queries each rule derivator in order of their priority. The EIA tries to improve the emergent solution to the set of recurring events using the exception rules from one derivator at a time. If the derivator cannot provide more rules, then the next derivator is queried. If there is no change in the emergent solution to the set of recurring events from previously attempted exception rules, then those rules are removed.

Author's personal copy

A rule derivator rd creates exception rules, $rd(sol^{\text{rec}}_{\text{opt}}, sol^{\text{rec}}_{\text{emg}}, n) = rules$, by comparing the estimated optimal solution $sol^{\text{rec}}_{\text{opt}}$ with the computed emergent solution $sol^{\text{rec}}_{\text{emg}}$ to the set of recurring events. A rule derivator returns $rules$ that attempt to address a single deviation between the emergent solution and the estimated optimal solution, that is a single location j with either $ev^{\text{rec}}_{\text{opt},j} \neq ev^{\text{rec}}_{\text{emg},j}$ or $Ag^{\text{rec}}_{\text{opt},j} \neq Ag^{\text{rec}}_{\text{emg},j}$.

The variable n tells the rule derivator to return exception rules for the nth (first, second, third, …) of these locations. This allows a rule derivator to search for exception rules to address later deviations even if no rules could be found to fix a previous one. If there exists no nth location, then the empty set is returned.

4.2.6 Send action

The objective of **send**(Ag_i, R_i), is to communicate the set R_i of exception rules to the agent Ag_i, the next time communication with Ag_i is non-disruptive. The agent will receive the exception rules and stores them in an internal data structure to be incorporated into its decision mechanism. The decision function of the agent determines which rules apply to its current state and environment. If no rules apply, then it performs the action it normally would. If multiple rules apply, a conflict resolution mechanism decides which rule or rules to apply. This conflict resolution is application dependent.

4.3 Exception rules

The EIA adapts the agents in the client multi-agent system by advising exception rules when consulted. An exception rule is a compositional adaptation designed to improve the efficiency of the system for a problem with a recurring set of events. Agents must be capable of processing situational information and determining if an exception rule needs to be applied. Exception rules influence an agent's decision making process by changing the decision function such that if no exception rule applies, then the regular action is taken. Otherwise, conflict resolution incorporating the rules' effects is used to determine the action taken.

When formulating exception rules the underlying coordination mechanism of the advised multi-agent system must be taken into account. Currently, three classes of exception rules have been considered [25] (see Fig. 3): event-triggered rules, time-triggered rules, and neighborhood-triggered rules. Event-triggered rules become activated by the perception of an event, time-triggered rules become activated at a certain point in time, and neighborhood-triggered rules become activated by the behavior of the other agents in the observed local environment. Two sub-types, ignore exception rules and pro-active exception rules, have already been explored.

Event-triggered ignore exception rules instruct an agent to ignore a particular abstract event for a period of time [25,28,49,50]. The hope is that another agent will perform the task resulting in a more efficient emergent solution. More precisely, an ignore exception rule is created for an agent from the emergent solution to the set of recurring events instructing it to ignore an event it originally completed.

Time-triggered/event-triggered forecast rules, also known as pro-active exception rules, influence an agent to begin work on an abstract event before this event usually appears [51,52]. This variant of exception rules has two purposes: (1) the rule-applying agent will be able to complete preparation for the specified event before it appears, resulting in quicker satisfaction of the event's requirements, and (2) an agent occupied with a pro-active exception rule will not be distracted by other events, hopefully leaving them for more suitable agents. Optimistically pro-active exception rules create a more efficient emergent solution. More precisely, a pro-

Author's personal copy

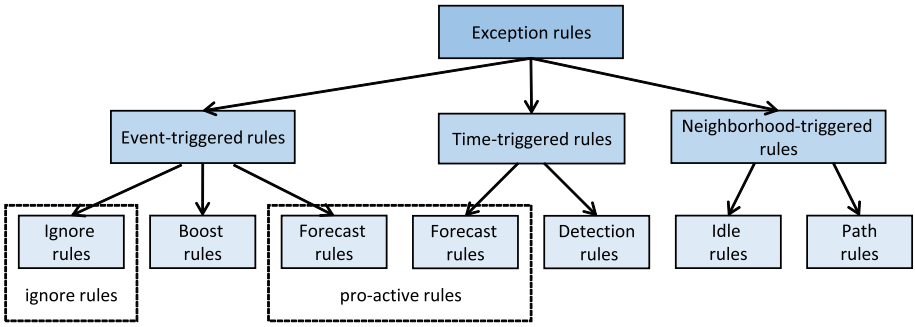


Fig. 3 Exception rule types

active exception rule is created for an agent from the estimated optimal solution to the set of recurring events instructing it to perform the preparation for the associated estimated optimal event on the occurrence of a trigger.

Depending on the application area some rule-types, such as ignore exception rules, may be free of conflicts [18]. However, the combination of rule-types, such as the addition of pro-active rules, can lead to unwanted emergent behaviors [52]. The base priority method used by the EIA to derive and apply exception rules begins with less restrictive rules, like ignore exception rules, and then moves on to pro-active exception rules. However, research has exposed that provided exception rules, despite being well intentioned, can have undesirable unintended consequences [52]. Consequently, it is important to assess and manage the risks associated with combinations of derived exception rules before they are sent to the agents of the client multi-agent system. This is the purpose of the RA-EIA introduced in Sect. 6.

5 Evolutionary learning of event sequences (ELES)

In this section, the exploratory testing method termed Evolutionary Learning of Event Sequences (ELES) is described. This testing method was originally developed in [17,18,52] as a human-managed genetic algorithm method of testing complex systems for emergent misbehavior. Emergence is unpredictable and, when combined with a large space of possible problem instances, assessment of associated risks without assistance is a seemingly overwhelming task.

ELES-assisted human-directed testing has been successfully used to demonstrate the efficiency losses of a base self-organizing emergent multi-agent system [16–18] as well as the potential gain from the use of a self-adaptive component such as the EIA [16,19,52]. The method has also been used to compare and contrast different exception rule types for the EIA [51,52].

ELES is used in two different contexts in this paper:

1. ELES is used in the human-directed evaluations of Sect. 8 to compare and contrast the risks associated with allowing a self-adaptive component such as the EIA/RA-EIA to adapt a self-organizing emergent multi-agent system. This consists of testing both the risk that the adaptation leads to a loss of efficiency in regards to the recurring set of events, and the risk that exploitation of an adaptation will lead to a severe loss of efficiency.

Author's personal copy

2. Second, as mentioned previously, in Sect. 6 the ELES method is automated and internalized in a resource-aware manner into the RA-EIA to enable the pro-active assessment of the risk of exploitation from a prospective rule.

The following sections will introduce the basic ELES genetic algorithm, as well as the two instantiations of the methodology for the risk assessment goals of this paper's human-directed evaluations.

5.1 Basic ELES methodology

Evolutionary algorithms are a search technique which sacrifice the promise of optimality in exchange for finding a good solution in an acceptable time period. These algorithms may be used to explore the extent of risk from prospective adaptations as well as for exploratory testing a self-adapting self-organizing emergent multi-agent system.

In this section, we define the process of using a genetic algorithm for searching within DTF problems. The first step of successfully implementing an evolutionary algorithm for search is to determine the representation of an individual. From the definitions of a run sequence, run instance, and event we can expand a run sequence of a DTF problem to a more descriptive presentation:

$$rs = (ri_1, \ldots, ri_k) = \begin{pmatrix} (ev_{1,1}, \ldots, ev_{1,m_1}), \\ \ldots, \\ (ev_{k,1}, \ldots, ev_{k,m_k}) \end{pmatrix} = \begin{pmatrix} ((ta_{1,1}, t^{\text{avail}}_{ta_{1,1}}), \ldots, (ta_{1,m_1}, t^{\text{avail}}_{ta_{1,m_1}})), \\ \ldots, \\ ((ta_{k,1}, t^{\text{avail}}_{ta_{k,1}}), \ldots, (ta_{k,m_k}, t^{\text{avail}}_{ta_{k,m_k}})) \end{pmatrix}$$

This can be simplified into the following representation of an individual as a sequence of tasks and associated times

$$(ta_1, t_1, \ldots, ta_p, t_p)$$

where $ta_i \in T$ and $t_i \in Time$.

The process of testing begins with an initial population of random individuals. Each individual in this population is simulated and its performance is evaluated and assigned a fitness value using a provided testing goal-specific fitness function. Subsequent populations are formed by using the selection process sel_{proc} to choose individuals from the current population and creating new individuals using genetic operators. This new population is subsequently simulated which starts this process again. This generally continues until a number of generations (count of populations) have passed.

Genetic operators are used to accomplish a transition between population states. A genetic operator is a function which uses a selection process $sel_{\text{proc}}(pop)$ to choose some number of individuals from the population pop which are altered and combined to form a new individual $indi$. Generally sel_{proc} is biased towards selecting individuals that are the most fit. Most methods are a variation of fitness proportionate selection where more fit individuals are more likely to be selected, but there is no guarantee. Essentially they are a random selection with weightings that are biased towards returning more fit individuals. Other methods include, elitist selection, roulette-wheel selection, and rank selection. The following are instantiations of standard genetic operators for DTF.

- A best operator $Best_{\text{Op}}$ is a genetic operator that selects the most fit individual to remain in subsequent populations.
- A survival operator $Surv_{\text{Op}}$ is a genetic operator that selects one of the most fit individuals to survive into subsequent populations.

Author's personal copy

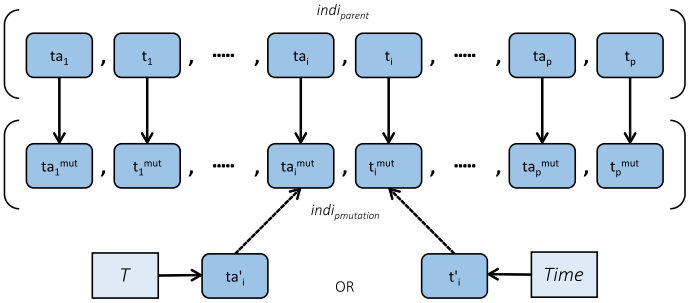


Fig. 4 Mutation operator

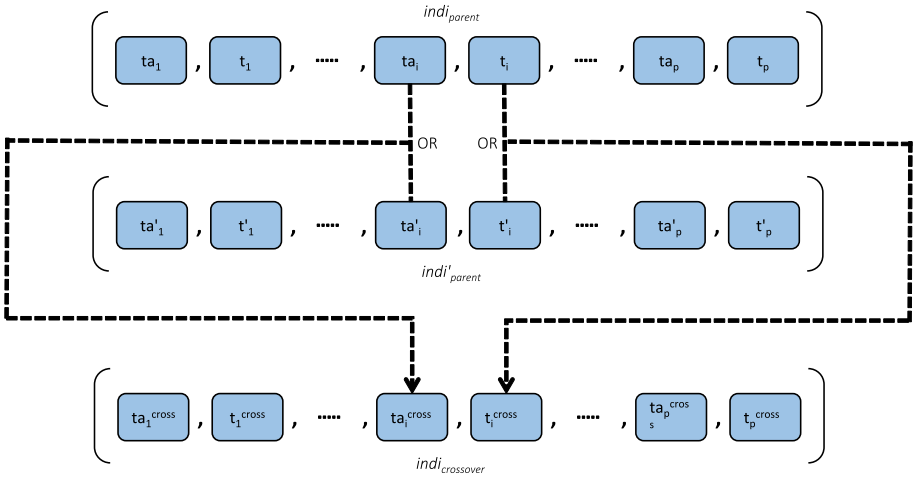


Fig. 5 Crossover operator

- A mutation operator $Mut_{Op}(indi_{\text{parent}}) = indi_{\text{mutation}}$, (see Fig. 4), is a genetic operator where a single parent individual $indi_{\text{parent}}$ is selected and a single $ta_i \in T$ or $t_i \in Time$ in that individual is mutated to another $ta'_i \in T$ or $t'_i \in Time$ to create a child individual.
- A crossover operator $Cross_{Op}(indi_{\text{parent}}, indi'_{\text{parent}}) = indi_{\text{crossover}}$, (see Fig. 5), is a genetic operator where two parent individuals $indi_{\text{parent}}$ and $indi'_{\text{parent}}$ are selected and a fair coin chooses either $ta_i \in T$ or $ta'_i \in T$ and either $t_i \in Time$ or $t'_i \in Time$ for each i to create a child individual.

There are three steps to tailor the general ELES method to a specific exploratory testing goal:

1. A determination of what form of run sequence is represented by the individual.
2. A determination of how to quantify the fitness of an individual is required.
3. Additional genetic operators, which are termed targeted operators, may be added to better guide the evolutionary search.

The following sections introduce two tailorings of the ELES method to the testing goals of the experimental evaluations completed in this paper. The example described across Sects. 8.2 and 8.4.1 is the result of the second tailoring of the ELES method described in the following sections.

Author's personal copy

5.2 Risk of performance loss from self-adaptation

The goal of this testing problem is to explore the potential that the adaptation of the system leads to a loss of efficiency in regards to the recurring set of events. More specifically, we want to design ELES to produce examples of run instances where the EIA examines the recurring set of events and advises exception rules that result in an undesirable drop in system performance efficiency. To accomplish this the system will be exposed to the run instance of an individual some k times to allow the EIA to adapt the system. We will follow the three steps to tailor the ELES method to this goal:

1. The run sequence rs^{risk} experienced by the system consists of k duplications of the run instance ri^{risk}.

   $$rs^{\text{risk}} = (ri_1^{\text{risk}}, ri_2^{\text{risk}}, \ldots, ri_k^{\text{risk}})$$

   This run instance ri^{risk} forms an individual solution in the population of the genetic algorithm.

   $$ri^{\text{risk}} = (ta_1, t_1, ta_2, t_2, \ldots, ta_p, t_p) = indi^{\text{risk}}$$

2. The primary part of the fitness function measures the decrease in emergent solution performance before and after adaptation. Note, this fitness function is based on inefficiency as a greater value of cost for the solution. Therefore, the goal of the fitness function is a solution in which the value for ri_k^{risk} is greater than that of ri_1^{risk}.

   $$prim(ri^{\text{risk}}) = MAX(0, qual(sol_{\text{emg}}(ri_k^{\text{risk}})) - qual(sol_{\text{emg}}(ri_1^{\text{risk}})))$$

   A secondary fitness function measures how much improvement remains after adaptation. This allows the quantification of situations in which performance may not have been decreased, but at least was not increased by much. This secondary measure is a comparison of the actual emergent solution after adaptation relative to the estimated optimal.

   $$sec(ri^{\text{risk}}) = MAX(0, qual(sol_{\text{emg}}(ri_k^{\text{risk}})) - qual(sol_{\text{opt}}(ri^{\text{risk}})))$$

   Finally, the tertiary fitness function benefits individuals that have high potential for improvement via adaptation by comparing the actual emergent solution before adaptation relative to the estimated optimal.

   $$tert(ri^{\text{risk}}) = MAX(0, qual(sol_{\text{emg}}(ri_1^{\text{risk}})) - qual(sol_{\text{opt}}(ri^{\text{risk}})))$$

   The sum is then normalized against the solution quality of the estimated optimal solution to form the fitness function:

   $$fit^{\text{risk}}(indi^{\text{risk}}) = \frac{\omega \cdot prim(ri^{\text{risk}}) + \upsilon \cdot sec(ri^{\text{risk}}) + \tau \cdot tert(ri^{\text{risk}})}{qual(sol_{\text{opt}}(ri^{\text{risk}}))}$$

   where ω is the primary weight factor, υ a secondary weight factor, and τ a tertiary weight factor. The weight factors should be chosen such that examples fulfilling the primary portion will be valued greater than those that only fulfill the secondary, and similarly for the primary and secondary portions over the tertiary. Therefore, in practice the primary weighting should be something like an order of magnitude above the secondary, and the secondary an order of magnitude above the tertiary.
3. There are no targeted operators required for this testing goal.

Author's personal copy

5.3 Dynamic performance risk of self-adaptation

For this testing problem the goal is to explore the risk that exploitation of an adaptation will lead to a severe loss of efficiency. More accurately, we want to design ELES to produce examples where the EIA adapts the advised system to one set of recurring events, then the problem changes resulting in a undesirable drop in efficiency. This change simulates the result of the influence of a malicious adversary creating specific events or series of unfortunate events. To accomplish this the system will be exposed to a pair of run instances where the system is allowed to adapt to the first one and then subsequently encounters the second. The goal is that this dynamic change results in unwanted behavior while solving the second run instance due to adaptations created for the first run instance.

An example of the pair of run instances produced by this ELES design is found in the experimental evaluations. Section 8.2 describes the first of the run instances the system positively adapts to via advised exception rules. Section 8.4.1 follows with the second run instance of the pair where the exception rules produce unwanted behavior as a result of the exception rules.

We tailor the ELES method to this goal via the following:

1. The run sequence rs^{break} consists of a pair of run instances ri^{adapt} and ri^{break}. For evaluation, the run instance ri^{adapt} is expanded to k duplications, which during simulation will be preceded and followed by ri^{break} to determine performance before and after adaptation of ri^{adapt}.

$$rs^{\text{break}} = (ri^{\text{break}}_{\text{before}}, ri^{\text{adapt}}_{1}, ri^{\text{adapt}}_{2}, \ldots, ri^{\text{adapt}}_{k}, ri^{\text{break}}_{\text{after}})$$

   These two run instances form the individual in the population of the genetic algorithm.

$$\begin{aligned} ri^{\text{adapt}} &= (ta_1, t_1, ta_2, t_2, \ldots, ta_p, t_p) \\ ri^{\text{break}} &= (ta_{p+1}, t_{p+1}, ta_{p+2}, t_{p+2}, \ldots, ta_{p+m}, t_{p+m}) \\ indi^{\text{break}} &= (ri^{\text{adapt}}, ri^{\text{break}}) \end{aligned}$$

2. The goal of the fitness function fit^{break} is to produce run sequences in which the adaptation of the system to encountering the run instance ri^{adapt} results in a decrease in performance for the run instance ri^{break} compared to if the system had not adapted. This goal is evaluated by the primary $prim$ portion of the fitness function, while the secondary and tertiary portions of the fitness function serve to guide the fitness function to eventually produce examples for the primary goal.
   To accomplish this the fitness function primarily determines the decrease in the system's emergent solution performance for the run instance ri^{break} from before to after adaptation.

$$prim(ri^{\text{break}}) = MAX(0, qual(sol_{\text{emg}}(ri^{\text{break}}_{\text{after}})) - qual(sol_{\text{emg}}(ri^{\text{break}}_{\text{before}})))$$

   To guide the fitness function towards instances in which the system created beneficial adaptations to ri^{adapt}, a secondary part of the fitness function is used. The secondary portion measures how improved the system's performance was for the ri^{adapt} by comparing the solution qualities of the actual emergent solution to the run instance ri^{adapt} before adaptation to after adaptation.

$$sec(ri^{\text{adapt}}) = MAX(0, qual(sol_{\text{emg}}(ri^{\text{adapt}}_{1})) - qual(sol_{\text{emg}}(ri^{\text{adapt}}_{k})))$$

   Lastly, the tertiary fitness function benefits individuals that have high potential for improvement via adaptation by comparing the actual emergent solution to the run instance

Author's personal copy

ri^{break} before adaptation relative to the estimated optimal solution possible.

$$tert(ri^{\text{break}}) = MAX(0, qual(sol_{\text{emg}}(ri^{\text{break}}_{\text{before}})) - qual(sol_{\text{opt}}(ri^{\text{break}})))$$

The sum is then normalized against the estimated optimal solution to form the fitness function:

$$fit^{\text{break}}(indi^{\text{break}}) = \frac{\omega \cdot prim(ri^{\text{break}}) + \upsilon \cdot sec(ri^{\text{adapt}}) + \tau \cdot tert(ri^{\text{break}})}{qual(sol_{\text{opt}}(ri^{\text{break}}))}$$

where ω is the primary weight factor, υ a secondary weight factor, and τ a tertiary weight factor. Again, the weight factors are chosen to value the primary portion over the other two and the secondary over the tertiary portion.

3. This exploratory testing instantiation will be assisted with a targeted operator utilizing the knowledge that the tested system creates adaptations triggered by specific events. Therefore, there is a high likelihood that emergent misbehavior from adaptation will result from adaptations created for events in ri^{adapt} being triggered by similar events in ri^{break}. Subsequently, the evolutionary process is biased towards creating similar events in the pair of run instances. In general this consists of making the time of the tasks' occurrences and general properties similar.
   This targeted genetic operator is called the twin mutation operator $Twin_{\text{Op}}$. It selects from a single parent individual $indi_{\text{parent}}$ a $ta_i \in T$ and $t_i \in Time$ pair selected from the start portion of the individual representing ri^{adapt} and a $ta_{p+j} \in T$ and $t_{p+j} \in Time$ pair selected from the end portion of the individual representing ri^{break}. These two task-time pairs are replaced with new $ta'_i, ta'_{p+j} \in T$ and $t'_i, t'_{p+j} \in Time$, such that ta'_i is similar to ta'_{p+j} and t'_i is similar to t'_{p+j}, to produce a child individual.

6 Risk-aware efficiency improvement advisor (RA-EIA)

Now that both the general concept of the EIA and ELES have been described, this section will explain the structure of a Risk-Aware EIA (RA-EIA) which expands upon the EIA to assess and manage the risks of proposed adaptations to a self-organizing emergent multi-agent system. This risk assessment is accomplished via:

1. MCS to assess the regular operation risks of expected emergent misbehavior.
2. ELES to assess the severity of risk that may result from exploitation of the system by the malicious interference of an adversary creating specific events.

Automated management of these assessed risks enables the creation of a trustworthy reliable system capable of independent operation over extended periods. The initial and incomplete concept of the RA-EIA was introduced in [19] and was designed around assessing and managing only the risk of regular operational misbehavior through MCS. In this paper, this initial framework has been expanded to assess the severity of the extent of possible emergent misbehavior by integrating and automating the ELES method.

The combination of these two assessment methods allows the advised autonomous (self-directed) system to be trusted to achieve reliable autonomicity (self-management) in regards to efficiency. An example of the RA-EIA identifying and avoiding the risk of an exploited adaptation is described in Sect. 8.4.1 for the application area of PDP. First, Sect. 6.1 will examine the characteristics of the problem of risk assessment and management for a self-adapting self-organizing emergent system and the reasoning for the chosen solution. Section

Author's personal copy

6.2 follows with the abstract model, and Sect. 6.3 provides details relating to the introduced actions.

6.1 Problem

The challenge addressed by the RA-EIA concept developed in this paper is to manage the risks of EIA-proposed adaptations. More precisely, we want to make risk-aware decisions on different sets of proposed adaptations. We must accept a certain level of inefficiency given the uncontrollable nature of self-organizing emergent systems and the dynamic problems they solve. Therefore, we want to improve efficiency, while avoiding introducing possible inefficiencies, in regards to the original system state. Resultantly, the RA-EIA examines both the variability around the expected loss of a set of proposed adaptations, and the extent of expected loss given possible intentional or accidental abuse of the system.

Additionally, there are three key behaviors in the original EIA's management of adaptations that need to be addressed: (1) the EIA only determines a single exception rule at a time to address the first deviation between the emergent and estimated optimal solution, (2) the EIA requires feedback from the adapted real world system to determine the impact of an exception rule, and (3) the EIA continues advising rules as long as the emergent solution performance is worse than the determined optimal solution.

This means the advised system requires numerous, frequent, and possibly costly consultations with the EIA to achieve the critical number of exception rule adaptations to effectively improve performance. This reduces the autonomy of the advised system. It also introduces the possibility that good-intentioned, yet performance reducing, rules are inadvertently advised [52]. As well, the process often results in the advising of an excessive number of exception rules which restricts the behavior of the self-organizing emergent multi-agent system.

The RA-EIA addresses these problematic behaviors by introducing reflection into the self-adaptive control loop of the EIA component to form the introspective RA-EIA. To accomplish reflective introspection the RA-EIA uses predictive performance evaluations to assess efficiency risks under different prospective exception rules. Resultantly, during a single consultation, the RA-EIA accumulates a minimal set of risk-averse exception rules to advise.

The RA-EIA is designed to solve two important challenges related to assessing of the risk of emergent misbehavior.

1. Reducing the Frequency of Expected Emergent Misbehavior The behavior of a self-organizing emergent multi-agent system is unavoidably unpredictable as it manifests from the combined interactions of the system's agents. The dynamic nature of encountered problems means it is impossible to completely control the behavior of the system. A user must expect occasions where behavior fails to meet desired expectations. The predicted future efficiency cannot be determined from a single measurement and correct assessment requires a statistically significant sample [22].
2. Reducing the Severity of the Extent of Emergent Behavior
   Self-organizing systems are designed to deal with unexpected changes and should accomplish the system's goal given challenging situations [31]. It is possible and even likely, depending on the application, that an adversary exists with the desire to manipulate the problem. This malicious adversary can be considered to bias the problem occurrence to make near-to worst case problems more frequent. Finding near-to worst-case examples is computationally hard.

Consequently, the RA-EIA includes the ability to assess the risk of a proposed set of exception rules in two manners:

Author's personal copy

1. Assess the Frequency of Expected Emergent Misbehavior Using MCS
   Assessing the frequency of expected emergent misbehavior in order to reduce the introduction of future risks from adaptations requires accounting for the possible distribution of future problems. MCS allows the system to account for the variability of performance given a distribution of possible problems.
2. Assess the Severity of Emergent Misbehavior Using ELES
   Assessing the severity of worst-case behavior requires solving a search problem for examples of worst-case behavior that produce extreme emergent misbehavior. Intelligent search methods will sacrifice the deterministic guarantee of the optimal solution, but are necessary to produce adequate results in acceptable time. Evolutionary learning methods, such as ELES, allow the assessment of the extent of a costly outcome given malicious interference creating specific events.

The above methods enable the RA-EIA to forecast gain/loss outcomes of proposed exception rules. The RA-EIA achieves loss control, by choosing rules in order to reduce the frequency and severity of losses. Additionally, the RA-EIA's assessment quantification of severe risk allows for the owner of such a system to manage the associated liabilities through insurance. This risk assessment and management is accompanied with computation costs which are acceptable considering that the advice from a single consultation with the RA-EIA is designed to provide the client with long-term reliable efficiency improvement.

6.2 RA-EIA model

The RA-EIA improves upon the existing EIA concept. The derivation process of the EIA is naive in regards to the long-term consequences of the advised exception rules. The RA-EIA replaces the **derive** action of the original EIA concept with its own $\mathbf{derive}_{\mathrm{RA-EIA}}$. Within this action reflection is accomplished via predictive performance evaluation through MCS and ELES. The RA-EIA uses these risk assessment methods to determine a minimum set of effective, yet risk-averse, exception rules.

To enable this there are three additional requirements for the RA-EIA:

1. A statistical distribution of the expectation of future events must be producible by the RA-EIA to enable the application of MCS.
2. The RA-EIA requires an idea of the possibilities an adversary has to influence the events the system encounters in order to tailor the ELES search.
3. A prediction or simulation of the performance evaluation of the base self-organizing emergent system when encountering a DTF problem instance must be possible. Without prediction the required reflection is impossible and determining the consequences of rules would require a costly and time consuming real world implementation which negates many of the benefits of the method.

The RA-EIA currently does not have the ability to gather more information from agents during the derivation action. The RA-EIA waits until the desired history information is communicated with the advised system's agents before it begins the process of examining it. If information arrives during the rule derivation process, then this information is considered in the next iteration of the rule derivation process. More flexible methods of integrating information during operation are an interesting avenue of future research into the EIA/RA-EIA concept. If it is necessary to get more information during the derivation process because the recurring events have changed, then this may indicate that the RA-EIA may not be a suitable solution as it is unable to keep up with the changing problem.

Author's personal copy

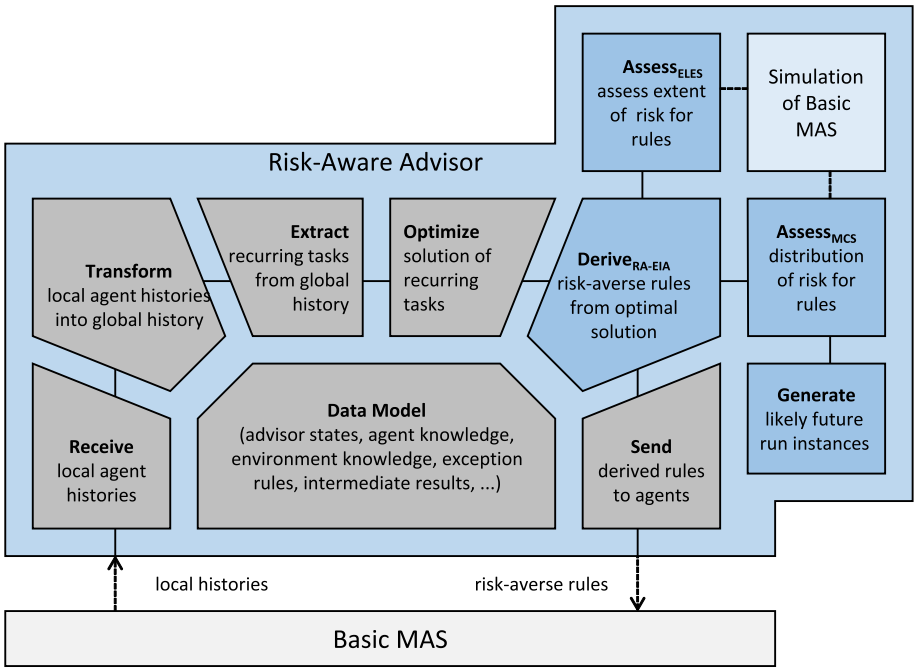


Fig. 6 Functional architecture of RA-EIA

The architecture of the RA-EIA is depicted in Fig. 6. The two sub-actions **assess**$_{\text{MCS}}$ and **assess**$_{\text{ELES}}$ are utilized by the **derive**$_{\text{RA-EIA}}$ action to assess risk. The **derive**$_{\text{RA-EIA}}$ action may use just MCS or both MCS and ELES. Each sub-action enables the assessment of a different measure of the risk. The **assess**$_{\text{MCS}}$ sub-action determines a mean for the overall level of average efficiency change relative to the default. The **assess**$_{\text{ELES}}$ sub-action determines the extent of inefficiency relative to the default.

6.3 RA-EIA actions

The communication/interaction between an agent Ag_i and the RA-EIA agent $Ag^{\text{RA-EIA}}$ is depicted in Fig. 7. It is important to note that the RA-EIA agent operates offline from the advised system taking the necessary time to determine risk-averse exception rule advice. This time depends on the problem complexity, the desired accuracy of the assessment, and whether one or both of the assessment methods are used. The new **derive**$_{\text{RA-EIA}}$ action is formed by the sub-actions: **assess**$_{\text{MCS}}$, **assess**$_{\text{ELES}}$, and **generate**.

6.3.1 RA-EIA derive action

The action **derive**$_{\text{RA-EIA}}(sol^{\text{rec}}_{\text{opt}}, sol^{\text{rec}}_{\text{emg}}, ri^{\text{rec}}, NR) = rules$, replaces the original **derive** of the EIA and creates for each agent Ag_i a minimal set R_i of risk-averse exception rules, where R_i can also be empty. The action utilizes reflective risk analysis during the derivation of exception rule advice through the usage of the sub-actions **assess**$_{\text{MCS}}$ and **assess**$_{\text{ELES}}$. The **derive**$_{\text{RA-EIA}}$ action consists of a loop until either the rule derivators no longer provide unique prospective exception rules, or the time given for deriving the exception rules expires. The

Author's personal copy

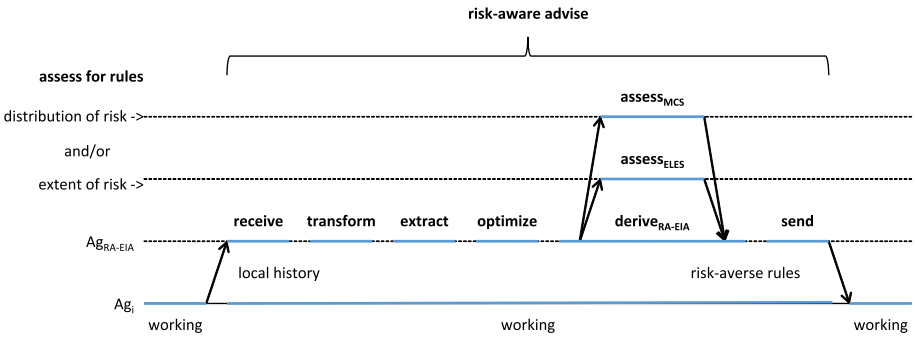


Fig. 7 Interaction between agent and RA-EIA

loop initially has an empty set $rules_{\text{acc}}$ of accumulated rules, and a set $rules_{\text{exist}}$ of currently advised rules. The rule derivators start by considering the first difference ($n := 1$) between the existing emergent solution and the estimated optimal solution to the set of recurring events.

The **derive**$_{\text{RA-EIA}}$ action's loop is as follows:

1. The RA-EIA utilizes the rule derivators to produce prospective exception rules for addition to the accumulated exception rules $rules_{\text{acc}}$.
2. – If there is no malicious adversary, then nothing needs to be done.
   – Otherwise, for each set of prospective exception rules $rules_i$, returned by a rule derivator rd_i, a measurement $y_i^{\max}$ is assessed. The measurement $y_i^{\max}$ is the maximum of multiple y_i measurements assessed using the **assess**$_{\text{ELES}}$ action for the rule-set $rules_{\text{exist,acc},i}$, which consists of the currently advised, the accumulated, and the prospective exception rules. If the measurement $y_i^{\max}$ exceeds the threshold $y_{\text{threshold}}$, then the prospective exception rules, $rules_i$, are removed from consideration.
3. For each set of prospective exception rules $rules_i$ remaining after the previous step, a sample mean $\overline{x_i}$ is assessed. This mean is assessed using the **assess**$_{\text{MCS}}$ action for the rule-set $rules_{\text{exist,acc},i}$, which consists of the currently advised, the accumulated, and the prospective exception rules. Additionally, the mean $\overline{x_{\text{acc}}}$ for the sample is assessed for only the accumulated rules using the **assess**$_{\text{MCS}}$ action for the rule-set $rules_{\text{exist,acc}}$ consisting of the rules currently advised and accumulated.
4. – If none of the prospective sets of exception rules produces a negative sample mean that improves on the accumulated set of exception rules (i.e. $\forall \overline{x_i} | \overline{x_i} \geq \overline{x_{\text{acc}}}$), then none of the prospective sets of exception rules provided a further improvement. Therefore, the rule derivators are incremented to look at the next difference $n := n + 1$.
   – Otherwise, if at least one sample mean is negative, then the exception rules associated with the smallest sample mean $x_{\min}$, which are $rules_{\min}$, are added to the accumulated set $rules_{\text{acc}}$. Additionally, the rule derivators are then reset to look at the first difference $n := 1$.

The **derive**$_{\text{RA-EIA}}$ action ultimately produces a set $rules_{\text{acc}}$ of accumulated exception rules determined to provide the greatest increase of average performance relative to the unadvised system. If the **assess**$_{\text{ELES}}$ sub-action was used, then the extent of performance risk for $rules_{\text{acc}}$ is assessed to be under the threshold $y_{\text{threshold}}$.

6.3.2 Assess MCS sub-action

The sub-action **assess**$_{\text{MCS}}(ri^{\text{rec}}, NR, rules) = \overline{x}$, uses MCS to determine a sample mean $\overline{x}$ for the distribution of the simulated performance for the proposed rules $rules$ compared to

Author's personal copy

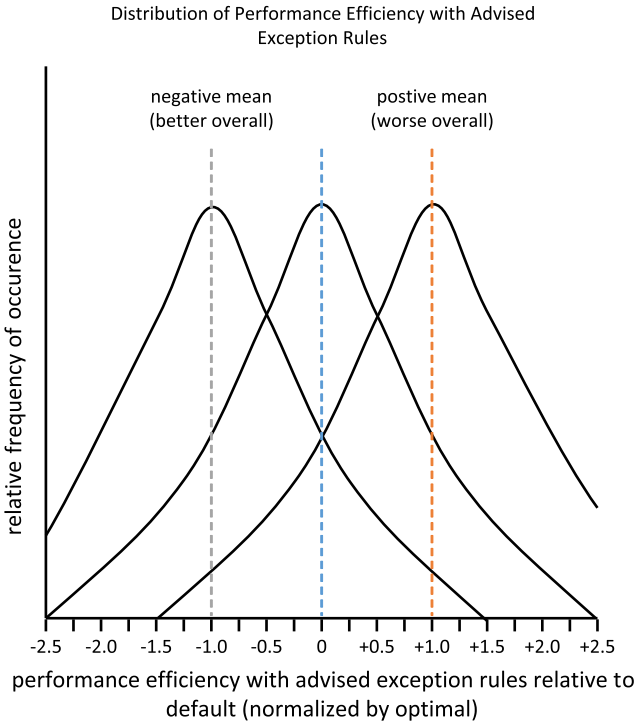


Fig. 8 Comparing $assess_{\text{MCS}}$ means

the default performance. A negative mean is a gain while a positive mean is a loss (see Fig. 8). The more negative the mean of a set of prospective rules, the larger the overall efficiency gain.

The action performs MCS using the **generate** action to create possible future run instances. These generated run instances are the inputs x^{input} for the stochastic MCS models f and f^{rules}. The model f is a simulation of the default self-organizing emergent multi-agent system without exception rules while f^{rules} is the same model after being advised by the exception rules $rules$. These run instances are generated until a statistically acceptable amount n_{req} has been created such that the sample mean $\overline{x}$ from the Y^{output} is acceptable or a maximum $n_{\max}$ is reached. A depiction of a stochastic MCS model can be seen in Fig. 9.

To determine if a sample set is sufficient for mean estimation from the distribution we will use the following calculation to determine the number of measurements required n_{req} (i.e. sample size) [39].

$$n_{\text{req}} = \frac{z_\alpha^2 \cdot \sigma^2}{\delta^2}$$

This measurement uses a chosen level δ of absolute error. This is the desired accuracy of the mean. An engineering estimate of $range(Y_i)/4$ is used after a minimum count $n_{\min}$ of measurements have been taken since the actual population deviation σ is unknown. This is known as the range rule of thumb. The z-score z_α of the chosen confidence level α determines the number of measurements required to trust the estimation of the mean within the absolute error.

Author's personal copy

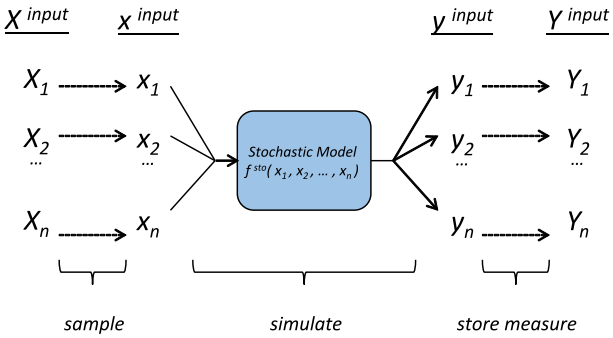


Fig. 9 Stochastic Monte Carlo simulation model

To form a sample, each run instance generated is simulated through the default and adapted models of the tested system and evaluated for a measurement:

$$y_{\text{output}} = \frac{(qual(sol_{\text{emg}}^{\text{after}}(ri^{\text{gen}})) - qual(sol_{\text{emg}}^{\text{default}}(ri^{\text{gen}})))}{qual(sol_{\text{opt}}(ri^{\text{gen}}))}$$

This measurement compares the solution quality of the system's emergent solution after being advised by the proposed set of exception rules to the emergent solution of the system without exception rules. This measurement is normalized by the solution quality of the estimated optimal. Negativity of a measurement implies the proposed exception rules improved the system's emergent solution relative to the unadapted base system. If the eventual sample mean $\overline{x}$ of the measurements Y^{output} collected is negative, then the prospective rules $rules$, on average, represent an overall efficiency improvement relative to the baseline (within a statistical confidence and absolute error).

6.3.3 Generate sub-action

The sub-action **generate**$(ri^{\text{rec}}, NR) = ri^{\text{gen}}$, creates a run instance ri^{gen} that is likely to occur in the future using the set ri^{rec} of recurring events and the set NR of non-recurring events extracted from the global history. This run instance is determined by examining the statistical distribution of the characteristics of previous run instances X^{input}. The number of events is chosen from the recent sizes seen in the last k run instances of the global history. The new run instance ri^{gen} begins as the existing set ri^{rec} of recurring events, and additional task-time event pairs are added from the set NR of non-recurring events. These events are chosen to not be duplicates of those already included in the generated run instance. Note, if there are no non-recurring events, then every run instance generated by this sub-action will be the set of recurring event.

6.3.4 Assess ELES sub-action

The sub-action **assess**$_{\text{ELES}}(ri^{\text{rec}}, NR, rules) = y$, uses ELES to measure the extent of the cost y, given the interference of an adversary, for the prospective rules $rules$ compared to the default performance. The measurement y is derived from the fitness of the run instance returned by the ELES genetic algorithm. The more positive the measurement y the larger the overall loss is relative to the default performance (see Fig. 10). A limiting threshold $y_{\text{threshold}}$

Author's personal copy

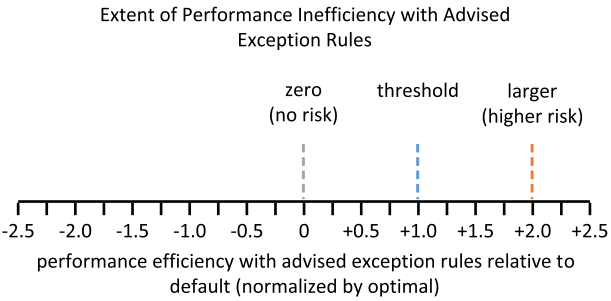


Fig. 10 Comparing $assess_{\text{ELES}}$ measurements

allows a designation of a point at which larger inefficiency measurements y are no longer acceptable.

This is an instantiation of the ELES method that was described in Sect. 5. However, in this instantiation the search space is limited by the extent the adversary may maliciously influence the occurrence of events. Therefore, the scope of the search space is for a good part determined from the global history and includes the number of tasks, the length of the period of $Time$, and the current exception rules of the system.

In order to simplify the explanation of the $\textbf{assess}_{\text{ELES}}$ sub-action we will describe it in terms of the three steps described in Sect. 5 for tailoring the general ELES method to a specific exploratory testing goal. The sub-action is concerned with comparing the system's already achieved adaptation along with the proposed exception rules to the default configuration. This allows for a threshold of severity of allowed emergent misbehavior from adaptations to be set.

The testing goal is therefore to produce run instances where using exception rules reduces efficiency to an unacceptable level compared to the default base system. To accomplish this the system will be exposed to two duplications of the run instance of an individual. The first run instance is simulated with the default configuration and the second with the prospective exception rules. We will follow the three steps to tailor the ELES method to this goal:

1. First, the run sequence rs^{assess} that the system will be exposed to consists of two duplicates of the run instance ri^{assess}.

$$rs^{\text{assess}} = (ri_1^{\text{assess}}, ri_2^{\text{assess}})$$

   This run instance ri^{assess} forms an individual solution in the population of the genetic algorithm.

$$ri^{\text{assess}} = (ta_1, t_1, ta_2, t_2, \ldots, ta_p, t_p) = indi^{\text{assess}}$$

2. Second, the fitness function fit^{assess} used to quantify the fitness of the individual $indi^{\text{assess}}$ is rather simple. Note, that the fitness function is essentially the same measure used in the $\textbf{assess}_{\text{MCS}}$ action for MCS. We will primarily consider the solution qualities of the actual emergent solution $sol_{\text{emg}}^{\text{after}}$ to the run instance after adaptation and the default emergent solution $sol_{\text{emg}}^{\text{default}}$ to the run instance without exception rules. We then normalize this value against the solution quality of the estimated optimal solution.

$$y = fit^{\text{assess}}(indi^{\text{assess}}) = \frac{(qual(sol_{\text{emg}}^{\text{after}}(ri^{\text{assess}})) - qual(sol_{\text{emg}}^{\text{default}}(ri^{\text{assess}})))}{qual(sol_{\text{opt}}(ri^{\text{assess}}))}$$

Author's personal copy

3. Third, a targeted operator $Rule_{\text{Op}}$ allows the ELES genetic algorithm to make use of the knowledge that for exception rules to influence ri^{assess} negatively it is likely that the evolved events will be similar to the events that trigger the exception rules. This process is similar to the regular ELES mutation operator. A single parent individual $indi_{\text{parent}}$ is selected by sel_{proc} and a single $ta_i \in T$ and $t_i \in Time$ pair will be selected and mutated to another $ta_i' \in T$ and $t_i' \in Time$. The important difference is that this mutation is derived from an existing exception rule.

7 Implementation for pickup and delivery problems

This section describes the instantiation of the abstract RA-EIA concept for the Pickup and Delivery Problems (PDP) subclass of DTF. This implementation for the application area of logistics is used in the experimental evaluations section to compare and contrast the EIA and RA-EIA. Section 7.1 defines PDP. Section 7.2 describes the Pollination Inspired Coordination (PIC) instantiation of a self-organizing emergent multi-agent system for PDP. Finally, Sect. 7.3 describes the details of the extensions required for the EIA/RA-EIA implementation. Each of these sections provides a detailed enough description for the purposes of this paper with further details available in the associated references for interested readers.

7.1 Pickup and delivery problems (PDP)

To evaluate our instantiation of a self-adapting self-organizing emergent multi-agent system for DTF we require a more concrete subclass of DTF, such as the NP-Hard PDP. DTF problems were originally defined in Sect. 2. PDP require the service of transportation requests (events) in a world (represented as a graph) during a period of time by a set of vehicles (agents), located at one or more depots. A solution consists of determining a set of routes for each vehicle which fulfills the transportation requests and associated constraints [54]. PDP are an unavoidable problem faced by logistics companies who are interested in flexible and efficient methods which reduce associated operational expenditures.

In the evaluation we are concerned with dynamic PDP consisting of transportation requests to move full-truck loads between stations S using a set V of vehicles. Full-truck-load problems indicate that only a single request may be completed by a vehicle at a time. In multi-vehicle-problems there is more than one vehicle in V. In dynamic problems, some information only becomes available within a limited time frame prior to the start of a request. A load must remain on one vehicle for the duration of its transport between paired pickup and delivery locations.

The set S of stations consists of pickup stations $PS \subseteq S$ and delivery stations $DS \subseteq S$ such that $S = PS \cup DS$. The set $S_0 = S \cup 0$ additionally includes the depot. A vehicle $v \in V$ has an initial location (the depot) and a capacity cap (maximum load size). The environment is a directed graph of discrete connected locations. A PDP instance is made up of transportation requests R where a transportation request $ev^{\text{PDP}} \in R$ is a task-time event pair $(ta^{\text{PDP}}, t_{ta^{\text{PDP}}}^{\text{avail}})$ where the task is $ta^{\text{PDP}} = (Prop_{\text{ta}^{\text{PDP}}}, [t_{\text{ta}^{\text{PDP}}}^{\text{start}}, t_{\text{ta}^{\text{PDP}}}^{\text{end}}])$ and properties of the task are $Prop_{ta^{\text{PDP}}} = (ps, ds, ls)$. Note, $ps \in PS$ is a pickup station, $ds \in DS$ is a delivery station, and ls is the load size.

A solution to a PDP instance consists of the assignment of the vehicles in V to transportation requests in R such that: every vehicle starts a route at the depot and returns to the depot at the end, every vehicle's capacity is not exceeded throughout its tour, a pickup and its associated delivery are served by the same vehicle, a pickup is always made before its

Author's personal copy

associated delivery, and any additional objectives are fulfilled. The goal of the system is to minimize the total distance traveled by all agents in A to produce the solution.

7.2 Pollination-inspired coordination (PIC)

To instantiate a self-organizing emergent multi-agent system we required a coordination method for PDP. This is accomplished through Pollination-Inspired Coordination (PIC), an extension of Digital Infochemical Coordination (DIC) a self-organizing emergent multi-agent system solution. Given the complexity of DIC and PIC, the explanation of this paper will be brief and the papers [25–27,29,30] can be referenced for explicit details. The importance to this paper is that PIC provides the agents of the system with an autonomous self-organizing coordination method to solve PDP.

A (digital) infochemical conveys information between two entities. PIC is inspired by the coordination apparent in pollination. The design goal is a multi-agent system of reactive PDP agents (vehicles) that autonomously respond to transportation requests. The result is completely decentralized and a PDP instance is solved by the system's emergent behavior. The PIC system, due to its decentralized self-organizing nature, is highly scalable, flexible, robust, and capable of completing dynamic PDP. However, such a system cannot guarantee efficiency, controllability, or observability.

The reactive agents (vehicles) of the PIC multi-agent system are infochemical agents that are only aware of their local infochemical environment. Each agent is only aware, may only "*smell*", the infochemicals that are situated on its location in the graph. The infochemicals are propagated and evaporated across the connections to other locations. Agents complete transportation requests by following the gradients of released infochemicals.

Transportation requests are made available in the environment by the paired pickup station ps and delivery station ds locations emitting request chemicals. A request chemical indicates to an agent the properties of the PDP transportation request $Prop_{\text{ta}^{\text{PDP}}} = (ps, ds, ls)$, which includes the load size ls. The emitter agent continues to re-announce the request chemicals as long as it remains incomplete in order to counter the concentration loss due to evaporation over time.

A PDP agent determines its action according to a priority method. Agents carrying a load search for the request infochemical of the delivery station to follow. Otherwise, the agent evaluates the infochemicals at its location. If there is a request infochemical, then a utility is computed. This utility promotes the spatial proximity of transportation requests while demoting request infochemicals for which there is an associated infochemical released by other agents or stations. If there is a transportation request with a positive utility, then the request infochemical with the highest utility is followed. If there are no request infochemicals, then after a number of idle iterations the agent will follow infochemicals to return to the depot.

7.3 EIA/RA-EIA for PIC

The manner in which to create an EIA-assisted PIC multi-agent system is fully described in [25]. First the previous system's agents need to be able to apply rules and periodically communicate with the EIA agent for **receive** and **send** actions. The **transform** action uses the agents' actions, perceived infochemicals, and state to extract the transportation requests and stations. The knowledge of the vehicles and the environment allows for the creation of the global history. Next, **extract** determines the set of recurring events based on tasks being spatially close in time, distance, and load size via Sequential Leader Clustering [13]. The set of non-recurring events are the remaining tasks. The action **optimize** then determines the

Author's personal copy

estimated optimal solution to the now static set of recurring events via a genetic algorithm. This estimated optimal solution is compared to the emergent solution to determine if rule advice needs to be derived by **derive**.

There are two types of exception rules considered for this instantiation. First, a PDP ignore exception rule ir^{PDP}, as introduced in [25,49,50], advises an agent to ignore the infochemical indicating a specific task for a period of time. Ignore exception rules have the highest priority, negating any response to the indicated infochemical signal in an agent's decision function if they apply. The deficiency of PDP ignore exception rules is that they are unable to directly change the order in which an agent completes a request. Second, a PDP pro-active exception rule pr^{PDP}, as introduced in [51,52], instructs an agent to move to the location of a forthcoming PDP transportation request after the occurrence of a previous PDP transportation request from the estimated optimal solution. Pro-active exception rules are next in priority. As soon as one of these rules is found applicable it is acted upon. Pro-active exception rules supplement the ability of ignore exception rules with the ability to change the order of request completion for an agent. After applying applicable ignore exception rules, and if no pro-active exception rules apply, then the basic agent's decision function action is used.

The implementation of the RA-EIA is an improved extension of the implementation of the EIA. There are few specific changes required for its implementation. The $\textbf{derive}_{\text{RA-EIA}}$ action simulates the PIC self-organizing emergent multi-agent system under different proposed rules and produces statistics, most importantly a record of the solution quality of the emergent and estimated optimal solutions for each run instance. These are used by the $\textbf{assess}_{\text{MCS}}$ and $\textbf{assess}_{\text{ELES}}$ actions in determining the sample mean and extent of performance loss measurements respectively.

8 Experimental evaluation

This section presents a variety of experimental evaluations to demonstrate the RA-EIA's capabilities. More particularly, the EIA and RA-EIA are compared based on how they advise the PIC self-organizing multi-agent system for PDP described in the previous section. The evaluations use both experiments with randomly generated events for an unbiased comparison of the two advisors, and human-managed ELES exploratory testing experiments, as described in Sect. 5, for a fitness function biased comparison.

For reference, previously completed evaluations have already demonstrated the capabilities of the base self-organizing emergent multi-agent system to solve PDP, the EIA to improve efficiency of the base system through derived advice, and the different exception rule types for the EIA. Experimental evaluations have been successfully used to demonstrate the benefits as well as the efficiency losses of a base self-organizing emergent multi-agent system in [16–18,25,28]. The potential gain possible through use of the EIA as a self-adaptive component has been demonstrated in [16,19,25,28,49,50,52]. Additionally, the benefits and dangers of different exception rule types for the EIA have been examined in [51,52].

To begin, Sect. 8.1 will explain the general setup of the evaluations. This is followed, in Sect. 8.2, with an example of the EIA/RA-EIA improving the PIC self-organizing system using exception rule advice. This example is extended later in Sect. 8.4, to demonstrate the extent of risk from provided advice when the RA-EIA is only using MCS, and how this risk can be mitigated by the use of ELES in the RA-EIA's rule derivation.

Author's personal copy

The initial incomplete RA-EIA concept, introduced in [19], was designed around assessing and managing only the risk of regular operational misbehavior through MCS. The evaluations of this initial concept from [19] have been expanded upon in this paper with additional evaluations exploring its capabilities more thoroughly. These evaluations, as reported in Sect. 8.3, compare and contrast the EIA and RA-EIA's ability to improve efficiency when there is no malicious adversary (i.e. the RA-EIA uses only $\textbf{assess}_{\text{MCS}}$).

In this paper the initial concept for the RA-EIA has been expanded to address the severity of risk from exploitation, which is necessary to manage the complete range of types of risk. This more complete framework integrates and automates the ELES method from [18] in a resource-aware manner. Therefore, in Sect. 8.4 experimental evaluations demonstrating the utility of these additional capabilities are reported. These experiments examine the challenge represented by a malicious adversary creating extreme misbehavior through the introduction of specific events. As well, the experiments demonstrate the RA-EIA's ability to manage the corresponding risk with the addition of the $\textbf{assess}_{\text{ELES}}$ action. As mentioned, an example demonstrating this capability is described throughout this section.

Finally, in Sect. 8.5 the relative running time of the EIA and RA-EIA and resulting considerations are discussed.

8.1 Setup

The evaluations make use of two metrics: the total distance traveled by the agents and the number of exception rules created by the advisor. More precisely, we are concerned with comparing the emergent solution to the estimated optimal solution. Therefore, we will give efficiency measures using a formula for the normalized difference:

$$\frac{qual^{\text{PDP}}(sol_{\text{emg}}) - qual^{\text{PDP}}(sol_{\text{opt}})}{qual^{\text{PDP}}(sol_{\text{opt}})} = \text{Percent of Inefficiency}$$

The more rules advised the greater is the possible infringement on the flexibility of the self-organizing emergent multi-agent system and the more opportunities for a malicious adversary to exploit the system.

Experiments are completed for different combinations of recurring and non-recurring events. This is communicated via $xRyN$. For example $4R2N$ means there are 4 recurring and 2 non-recurring events. Across the experiments the non-recurring events are limited to at most half the amount of recurring events. The motivation of this limitation is that the EIA concept is unlikely to be applied to an applications without sufficient recurring events. The more noise (non-recurring events) in the system the less the EIA can improve performance. No performance gains can be made by the EIA for non-recurring events which have in the most part fixed unavoidable inefficiency costs. Therefore, the greater the number of non-recurring events the less relative value there is to reducing the costs associated with the recurring events.

The environment of the system is a 11×11 grid connected with bi-directional connections of length 1 or $\sqrt{2}$ for diagonals (see Fig. 11). A larger or smaller grid will result in longer or shorter distances traveled but not change the relative performance efficiency. There are 2 delivery vehicle agents, the number of recurring events is 4, 6, or 8, and the number of non-recurring events is between 0 and 4. The total complexity of events is therefore limited to at most 12 as the ELES method of exploratory testing used in the penultimate experiments requires many simulations of the advised system.

There are a number of ELES configuration values: each run of the evolutionary testing has 100 generations and a population of 25 individuals. The percent of individuals for each new

Author's personal copy

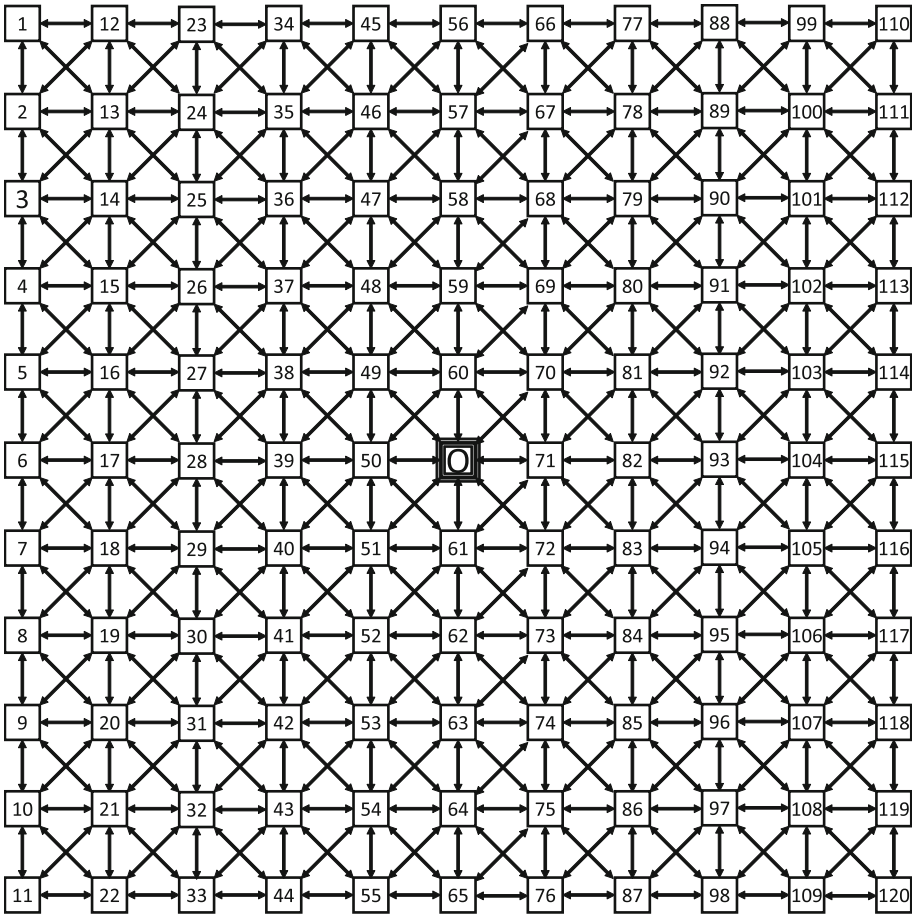


Fig. 11 Environment for self-organizing system

generation from each operator are: 10 % new random individuals, 5 % $Best_{\mathrm{Op}}$, 10 % $Surv_{\mathrm{Op}}$, 25 % $Cross_{\mathrm{Op}}$, and 50 % via mutation. This 50 % is split between the two mutation operators, Mut_{Op} and $Twin_{\mathrm{Op}}$, if both are used. The fitness function weightings are: 500 for primary, 10 for secondary, and 5 for tertiary. Each primary portion of the fitness function represents the actual desired result and thus is weighted much more than the other two portions. The secondary portion is more likely to influence the search towards examples that fulfill the primary portion so it is weighted twice as much as the tertiary.

For the **assess**$_{MCS}$ sub-action the confidence level is 95 % with an absolute error of 1 %. This is a common confidence level which allows for a sufficient level of accuracy without too much unnecessary additional computation with increasingly less return on investment. Less confidence and greater error can be exchanged in these parameters for reduced computation but less accuracy in the result. The **assess**$_{\mathrm{ELES}}$ completes 10 exploratory tests to determine the extent of risk. From the experimental results there was significant evidence that this many repetitions were not required to accomplish the desired result of the sub-action and that there are performance gains to be made around choosing the number of repetitions and generations of the ELES search.

Author's personal copy

Table 1 Adaptive run instance problem

Event	Pickup station	Delivery station	Iteration
A	86	75	23
B	24	39	26
C	21	119	33
D	29	69	54
E	120	30	57
F	67	48	92

8.2 Example of EIA/RA-EIA improvement

This section outlines a $6R0N$ example of EIA/RA-EIA adaptation to improve the PIC self-organizing system's emergent behavior for greater efficiency solving a set of recurring events. For the sake of clarity non-recurring events are not included in the example. This example will be utilized further on, in Sect. 8.4 of the experimental results, to demonstrate how the addition of specific non-recurring events can interact with provided exception rule advice to cause emergent misbehavior. Particularly, the extension shows how the use of ELES by the RA-EIA can prevent such extreme emergent misbehavior by identifying which exception rules carried unnecessary risk.

To begin, Table 1 describes a potential run instance problem that the self-organizing system may encounter. Table 1 contains a listing of the 6 recurring events that make up the run instance. Each of these events consists of a pickup station, a delivery station, and a time of occurrence. The delivery agents used by the self-organizing system to solve the problem can only perform one of these delivery requests at a time. The delivery agents begin at the central depot and are unaware of when or where the events for the current problem will occur. For clarity each of the events is given a letter coding along with a thicker border for each pickup station in the following figures.

Now, Table 2a plots the default behavior of the self-organizing system's response. In Fig. 12, a solid arrow path is used for the first agent Ag_1 and a dotted arrow path for the second agent Ag_2. Ag_1 responds to events $A - C - E - F$, in that order. Ag_1 attempts to return to the depot after the A delivery, partially responds to the B delivery after the C delivery until it perceives Ag_2 is responding to it, and also waits at the depot in between the E and F deliveries. Ag_2 responds to events $B - D$ and waits at the depot after delivering the B package until the D delivery appears. The total travel cost is ∼76.18.

There is quite a bit of room for improvement in the default PIC self-organizing system's response. The estimated optimal response possible, if the agents' were able to predict the future and see the global problem, is given in Table 2c. This estimated optimal response is determined by the EIA/RA-EIA by collecting the agents' local histories from a period of runs and extracting that the run instance in Table 1 was the recurring set of events. Given this recurring problem the EIA/RA-EIA uses a genetic algorithm to determine the estimated optimal response for the two available agents. In this case the estimated optimal is for one agent to perform a single loop of pickup and delivery without returning to the depot until the end. This loop also must be in a particular order which is different from the order in which the events appeared. The total travel costs of this estimated optimal path is ∼50.38.

There are four main sources of inefficiency demonstrated by the base self-organizing system in Fig. 12a relative to the estimated optimal:

1. Agents responding to an event another agent could have more efficiently performed. In the estimated optimal Ag_1 performs all the deliveries.

Author's personal copy

Table 2 Adaptive run instance solutions

Event	Vehicle	Order
(a) Default emergent solution		
A	1	1
B	2	2
C	1	3
D	2	4
E	1	5
F	1	6
(b) With advised rules		
A	1	1
B	2	2
C	1	3
E	1	4
D	1	5
F	1	6
(c) Estimated optimal		
A	1	1
C	1	2
E	1	3
D	1	4
F	1	5
B	1	6

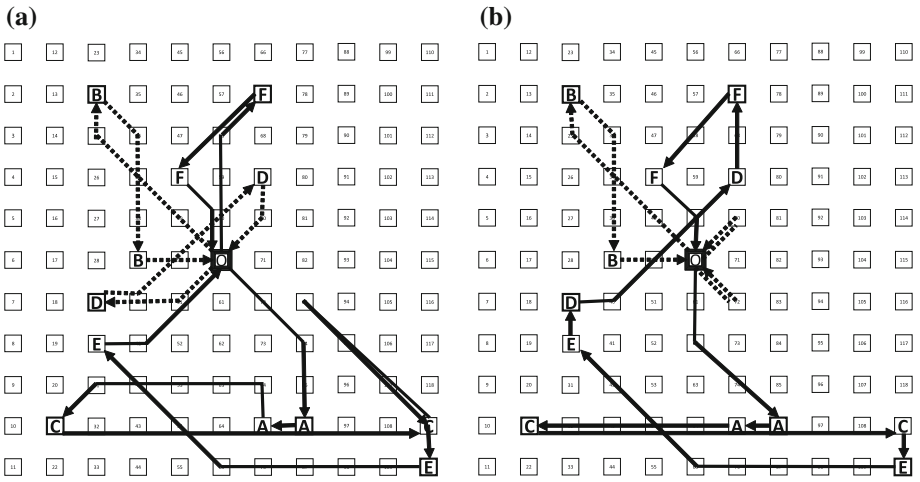


Fig. 12 Adaptive run instance solutions. **a** Default emergent solution, **b** with advised rules

2. Agents completing events in a sub-optimal ordering. In the estimated optimal events are not always performed in the order of their arrival. Particularly, the *B* delivery is delayed until the end.

Author's personal copy

3. Agents timeout waiting for a local delivery and return to the depot only to return to the same local area.
4. Agents partially responding to an event they don't complete. In the estimated optimal Ag_1 waits at the C delivery station for the E pickup station to become active.

The EIA/RA-EIA examines the differences between this estimated optimal solution and the system's current behavior and uses the provided rule derivators to produce targeted rules designed to influence the system's future behavior to be closer to the estimated optimal. In general, the first inefficiency of sub-optimal event assignment is addressed by ignore exception rules, and pro-active exception rules are used for the remaining three by effecting the ordering of completion and preparation of agents for future events.

The following four rules are an example of what exception rules the EAI/RA-EIA may produce:

- Agent 1
  - Pro-Active Rule 1—Move to station 21 after trigger of task appearing at station 86 on iteration 23 and timeout at iteration 33.
    Translation—After seeing A event move to and wait at location of C event until it should have appeared.
  - Pro-Active Rule 2—Move to station 120 after trigger of task appearing at station 21 on iteration 33 and timeout at iteration 57.
    Translation—After seeing C event move to and wait at location of E event until it should have appeared.
  - Pro-Active Rule 3—Move to station 67 after trigger of task appearing at station 29 on iteration 54 and timeout at iteration 92.
    Translation—After seeing D event move to and wait at location of F event until it should have appeared.
- Agent 2
  - Ignore Rule 1 - Ignore Task Appearing at station 29 on iteration 54 for 150 iterations
    Translation - Ignore D event.

These four rules are then communicated to the individual agents. The next time they encounter the run instance they will now respond as seen in Table 2b and Fig. 12b. The total travel cost is ∼60.87. Note, this response is not the optimal for two reasons. First, Ag_2 is partially responding to the E and F events before it perceives the infochemicals telling it Ag_1 is already responding to them. Second, the B task is still being completed by Ag_2.

The EIA/RA-EIA does not further advise the self-organizing system for two reasons. First, the two partial responses, which cost ∼5.66, are not captured in the emergent solution to the recurring events set of events extracted by the **extract** action which is used to judge if rule derivation should continue. Second, the response of Ag_2 versus Ag_1 adds two diagonal and horizontal segments which costs ∼4.83 in distance traveled. This additional amount falls within the $qual_{\text{threshold}}$ used to determine if rule derivation should continue.

This example demonstrates the impact of the EIA/RA-EIA providing exception rule advice. The key difference between the two advisors is that the EIA assumes the exception rules it derives will always result in positive improvements. The RA-EIA pessimistically assumes that exception rules can also have negative consequences, particularly due to the impact of non-recurring events. Therefore, the RA-EIA employs risk assessment via MCS to judge an exception rule on the basis of possible expected non-recurring events and via

Author's personal copy

ELES to judge the extent of exception rule misbehavior given specific targeted non-recurring events.

8.3 Expected variability of risk evaluations

This section will compare and contrast the original EIA concept with the risk aware RA-EIA variant. Note, for this section there is no malicious adversary creating targeted non-recurring events and therefore the RA-EIA will only assess risk via MCS.

The metric used in this section is efficiency improvement. It is based on the situation that the owner of a self-organizing emergent system must accept a certain level of unavoidable performance inefficiency due to the lack of controllability. The goal of the EIA is to improve efficiency relative to the base unadapted system. Furthermore, the goal of the RA-EIA is to reduce the risks associated with this improvement. Section 8.3.1 compares the two advisors using unbiased randomly generated examples. This is followed, in Sect. 8.3.2, by a comparison of the two advisors using human-managed ELES exploratory testing to determine the risk of performance efficiency loss during regular operation.

8.3.1 Random generated event sequences experiments

This section consists of two experiments consisting of randomly generated event sequences. In the first the non-recurring events are found in the global history and can be predicted by MCS, and in the second they are completely random. Each experiment consists of generating 50 run sequences for each combination $xRyN$ of events. Each run sequence is created by first randomly creating a set of x recurring events, then 30 run instances are generated by adding non-recurring events to the run instance. These y non-recurring events are either selected from a stable set of randomly generated non-recurring events or created randomly. For the first five run instances the global history is stored, then for the remaining run instances the advisors are given the opportunity to adapt the system.

For each run instance the following normalized difference metric of the percent of efficiency improvement of the advised system relative to the baseline system without advice is calculated:

$$\frac{qual^{\mathrm{PDP}}(sol_{\mathrm{emg}}^{\mathrm{base}}) - qual^{\mathrm{PDP}}(sol_{\mathrm{emg}}^{\mathrm{advisor}})}{qual^{\mathrm{PDP}}(sol_{\mathrm{opt}})} = \text{Percent of Improvement.}$$

The values produced for the final twenty-five run instances are averaged for the complete run sequence. The following tables report the min, max, mean, and standard deviation for the fifty run sequences. The $(+, -)$ rows allow a comparison between the two advisor variants, with positive values meaning the RA-EIA was better at improving efficiency and negative that the EIA was. A final row includes the statistical confidence that the means of the EIA and RA-EIA for a particular $xRyN$ combination are different. This statistical confidence was determined using the two-tailed t test [39]. Note, the values for test series with no non-recurring events (i.e. $4R0N$) are different between Table 3a and b because they are the result of two separate samples of the population possible for that series and do not consists of the same run sequences.

In Table 3a the non-recurring events are expected as they occurred in the first five run instances of the 30 that make up the run sequence. For many of the test series the EIA actually decreases the overall efficiency of the advised system compared to its default baseline. This is evident from min values for the EIA showing a negative. In contrast, the RA-EIA always produces non-negative min, which means that it consistently improves the system's

Table 3 Random performance

Advisor	Metric	Test series											
		4R0N (%)	4R1N (%)	4R2N (%)	6R0N (%)	6R1N (%)	6R2N (%)	6R3N (%)	8R0N (%)	8R1N (%)	8R2N (%)	8R3N (%)	8R4N (%)
(a) Non-recurring events expected													
EIA	Min	−18.81	0.68	3.13	−8.03	−2.23	−0.36	3.81	−8.62	−1.11	−4.74	−11.10	−1.76
	Max	76.75	51.60	36.37	82.09	49.05	40.17	33.44	48.04	50.67	28.02	26.96	23.58
	Mean	26.90	19.99	19.49	21.14	19.87	16.16	15.80	19.48	15.27	12.17	12.04	9.35
	Standard deviation	19.03	10.91	8.65	18.39	9.94	9.18	7.15	14.28	8.98	7.87	6.75	5.17
RA-EIA	Min	0.00	5.31	4.11	0.00	5.54	5.15	4.15	0.00	1.23	5.41	1.34	3.28
	Max	82.84	58.04	40.60	90.42	55.22	50.68	41.06	51.10	58.31	35.48	40.59	33.47
	Mean	23.62	23.77	21.52	27.09	26.19	21.67	19.95	26.51	24.15	19.30	17.63	13.63
	Standard deviation	18.82	11.51	9.43	17.13	10.17	12.03	8.44	12.90	11.25	7.80	7.61	6.60
(+/−)	Min	+18.81	+4.63	+0.98	+8.03	+7.77	+5.51	+0.34	+8.62	+2.34	+10.15	+12.45	+5.03
	Max	+6.09	+6.45	+4.23	+8.33	+6.17	+10.51	+7.63	+3.07	+7.64	+7.46	+13.63	+9.89
	Mean	−3.28	+3.79	+2.03	+5.95	+6.32	+5.51	+4.15	+7.03	+8.88	+7.13	+5.60	+4.29
	Confidence (mean)	61.16	90.53	73.40	90.28	99.78	98.84	99.07	99.88	99.99+	99.99+	99.98	99.95

Author's personal copy

Table 3 continued

Advisor	Metric	Test series											
		4R0N (%)	4R1N (%)	4R2N (%)	6R0N (%)	6R1N (%)	6R2N (%)	6R3N (%)	8R0N (%)	8R1N (%)	8R2N (%)	8R3N (%)	8R4N (%)
(b) Non-recurring events completely random													
EIA	Min	−8.37	−2.27	1.40	−23.85	−6.69	−0.39	−1.39	−5.49	−3.79	−6.32	−2.18	−1.37
	Max	63.77	47.45	35.92	60.84	55.26	31.45	34.70	44.45	57.79	23.65	22.84	18.59
	Mean	22.21	18.89	16.31	24.59	20.20	11.01	10.71	18.27	15.34	8.46	7.48	5.17
	Standard deviation	13.73	13.54	8.85	15.17	14.40	6.92	7.46	11.70	12.25	6.27	5.32	4.19
RA-EIA	Min	0.00	3.13	0.55	0.00	6.94	3.37	1.24	0.00	2.85	2.04	1.41	2.60
	Max	59.20	57.39	52.57	69.38	68.01	36.52	42.35	47.24	76.59	33.44	35.90	33.57
	Mean	19.14	23.50	21.80	28.85	27.74	20.28	17.71	23.85	25.72	17.88	15.16	12.89
	Standard deviation	13.79	13.87	11.30	16.45	13.58	7.82	8.91	12.38	13.67	7.09	7.75	6.87
(+/−)	Min	+8.37	+5.40	−0.85	+23.85	+13.63	+3.76	+2.63	+5.49	+6.63	+8.36	+3.59	+3.97
	Max	−4.58	+9.94	+16.65	+8.54	+12.76	+5.08	+7.66	+2.79	+18.80	+9.79	+13.07	+14.98
	Mean	−3.06	+4.61	+5.49	+4.25	+7.54	+9.27	+7.00	+5.58	+10.38	+9.42	+7.68	+7.73
	Confidence (mean)	73.19	90.41	99.18	81.80	99.17	99.99+	99.99+	97.74	99.99	99.99+	99.99+	99.99+

Author's personal copy

Author's personal copy

efficiency. The RA-EIA also improves the overall efficiency more than the EIA as indicated by the positive differential values (+, −).

In Table 3b the non-recurring events are completely random. The RA-EIA is subsequently unable to anticipate the future with MCS. However, the results still show that the RA-EIA on average outperforms the EIA. It can be determined from the examination of the logs related to these tests that this is accomplished by advising fewer rules and therefore a less restrictive set of rules. As in the previous experiment the RA-EIA avoids having any experimental test series with a negative min, unlike the EIA where almost all experimental test series do.

The null hypothesis for Table 3a and b is that there is no meaningful difference between the means produced by an experimental series for the EIA and RA-EIA. The confidence that the null hypothesis is false, i.e. that the means are unique and not a result of random chance, is reported in the final row of the table. It can be seen for $6R1N$ and above that this confidence is 90%+ and approaching 99%+.

Only for $4R0N$ did the EIA appear to perform on average better in the two tables. This is due to the EIA being constructed such that it will advise rules until it has forced the emergent solution into the estimated optimal or runs out of prospective rules. When there are only recurring events this means it advises rules until the emergent solution is the estimated optimal solution. These generally unnecessary extra exception rules are able to prevent unpredictable emergent misbehavior (which includes the partial response of agents to events they do not service). This unpredicted behavior is not captured by the solution quality $qual$. Without non-recurring events, or if the recurring events are static, this limitation of the system's flexibility does not harm the system performance. However, in the more likely case that there are non-recurring events and that the recurring events change over time, these additional rules are dangerous. These additional rules are one type of risk that the RA-EIA is designed to reduce.

8.3.2 ELES risk of performance loss experiments

The goal of this section is to determine the risks associated with the EIA and RA-EIA concepts through the use of experiments featuring ELES exploratory testing. These risks are the extent to which the addition of EIA and RA-EIA advice may negatively impact the performance of the advised system. This exploratory testing consists of twenty ELES experiments, ten for $4R0N$ and ten for $6R0N$. Five of each ten are for either the EIA or RA-EIA. Each experiment produces a single run instance as an example of the extent of performance efficiency loss for the system adapted by either advisor variant. See Sect. 5.2 for a description of the specific details for the ELES exploratory test genetic algorithm construction. These examples are then simulated five times and averaged to produce Table 4a and b reporting the baseline inefficiency along with the impact of the EIA/RA-EIA.

Reported is the baseline inefficiency before the default base system was adapted, along with the level to which the EIA and RA-EIA changed this inefficiency. Note, that the most fit run instance examples will be ones where the original baseline inefficiency is low and the advisor increases the inefficiency away from the estimated optimal solution at 0 %. Additionally, the min and max number of exception rules created in the 5 simulations are reported along with the number of rules created by each of the EIA and RA-EIA.

The first thing to note is that the ELES exploratory test experiments were unable to find a single example run instance for the RA-EIA fulfilling the primary portion of the fitness function fit_{risk} as a result of increasing the advised system's performance inefficiency. This means that the ten run instance examples for the RA-EIA only fulfill the secondary and tertiary

Author's personal copy

Table 4 ELES risk of performance loss

Test		Performance inefficiency			Exception rules			
Advisor	Series	Baseline (%)	EIA (%)	RA-EIA (%)	Min	Max	EIA	RA-EIA
(a) 4R0N								
EIA	1	9	111	9	0	5	5.0	0.0
	2	31	34	30	0	2	1.2	0.6
	3	25	276	25	0	4	4.0	0.0
	4	8	99	8	0	3	3.0	0.0
	5	69	104	40	0	5	3.8	1.8
RA-EIA	1	164	85	130	1	4	4.0	1.0
	2	183	183	183	0	0	0.0	0.0
	3	13	5	11	1	4	4.0	1.0
	4	51	20	10	1	4	4.0	3.4
	5	141	23	21	0	0	0.0	0.0
(b) 6R0N								
EIA	1	20	47	17	1	7	5.0	1.0
	2	18	51	16	1	6	5.6	1.0
	3	27	44	27	0	8	7.8	0.0
	4	38	36	32	0	8	6.0	0.2
	5	37	70	27	2	7	6.6	2.8
RA-EIA	1	192	86	122	1	5	5.0	1.4
	2	129	57	116	0	8	7.6	0.8
	3	132	20	52	0	7	6.6	3.4
	4	45	20	38	0	7	7.0	1.8
	5	115	23	21	1	9	7.2	2.6

portions. The secondary testing goal produces run instance examples in which improvement remains possible after adaptation. On the other hand, for the EIA the ELES exploratory testing produced run instance examples that did maximize the primary testing goal of the fitness function. The existence of these run instance examples confirms the previous speculation that, due to the lack of reflection when determining exception rules, the EIA is capable of unintentionally reducing the system's performance efficiency.

For the EIA examples, the average baseline inefficiency is ∼28 % in both tables. The EIA increased this inefficiency to on average ∼125 % for $4R0N$ in Table 4a and ∼50 % for $6R0N$ in Table 4b. In comparison, for the same EIA examples the RA-EIA avoided increasing inefficiency. On average the RA-EIA decreased the inefficiency to ∼22 % for $4R0N$ in Table 4a and ∼24 % for $6R0N$ in Table 4b. The RA-EIA accomplished this by either advising fewer exception rules or simply avoiding advising exception rules altogether when none could be found to decrease performance inefficiency.

The most important result is that the exploratory testing only produced examples fulfilling the primary portion of the fitness function, where inefficiency was increased by advised rules, for the EIA and not the RA-EIA. These examples are evidence that reflective risk assessment via MCS in the RA-EIA is an important addition to the EIA concept.

Author's personal copy

Table 5 Breaking run instance problem

Event	Pickup station	Delivery station	Iteration
Recurring events			
A	86	75	23
B	24	39	26
C	21	119	33
D	29	69	54
E	120	30	57
F	67	48	92
Non-recurring events			
$A*$	86	102	22
G	41	84	24

8.4 Extent of risk evaluations

This section of the experimental evaluation adds into consideration the existence of a malicious adversary and compares and contrasts the ability of the EIA and RA-EIA to handle the risks of exploitation. The adversary can be considered to either be a purposeful entity or simply represent unfortunate circumstance with the same effect. The malicious influence of this adversary is limited to the creation of events for the system to solve.

First, to demonstrate how advisor provided exception rule advice can be exploited by a malicious adversary Sect. 8.4.1 extends the example from Sect. 8.2. In this example, two particular non-recurring events are added to the run instance problem which interact with the advised exception rules to result in extreme emergent misbehavior. More specifically, the agents travel a much longer distance than before being advised. This example is concluded with a demonstration that this extent of misbehavior can be avoided by the RA-EIA identifying and avoiding one of the advised rules.

The example is followed by Sect. 8.4.2 which establishes that there are exploitation risks for both the EIA and the RA-EIA when the RA-EIA is only using the **assess**$_{\text{MCS}}$ sub-action. Finally, in Sect. 8.4.3, the RA-EIA utilizes the **assess**$_{\text{ELES}}$ sub-action in addition to **assess**$_{\text{MCS}}$ to counter the previously discovered situations, while still providing general efficiency improvements.

8.4.1 Example of dynamic performance risk

This section outlines a $6R2N$ example that is paired with the previous $6R0N$ example from Sect. 8.2. This example assumes the PIC self-organizing system has already been adapted to the recurring set of events by the EIA/RA-EIA communicating the four exception rules from the previous example. In this example the run instance is extended by two non-recurring events which interact with the advised rules to reduce the performance efficiency of the system greatly by subverting the previous positive impact of the exception rules. The example is concluded by showing that this misbehavior can be mitigated if some method, such as the RA-EIA using ELES, can identify and remove the exception rule causing the extreme emergent misbehavior.

To begin, Table 5 describes the extension of the previous run instance problem with two additional non-recurring events. One of the new non-recurring events shares a pickup location

Author's personal copy

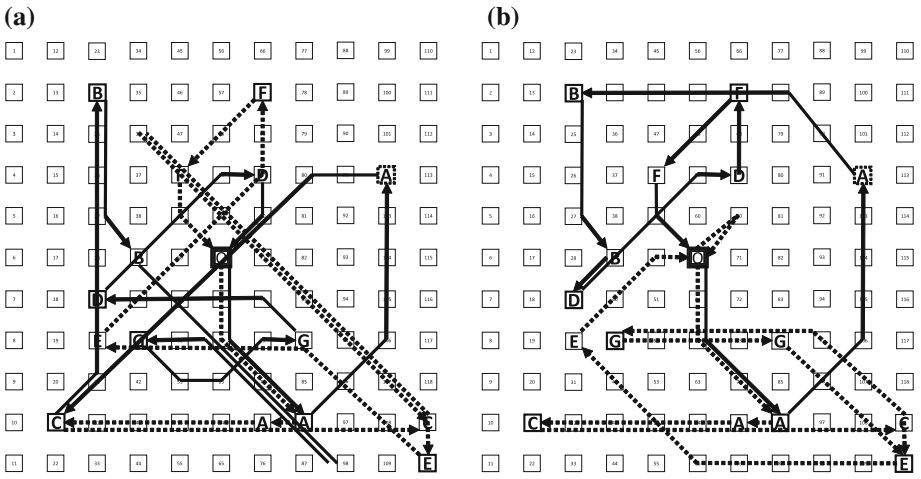


Fig. 13 Breaking run instance solutions. (a) Emergent solution with advised rules. (b) Minus pro-active rule 1

Table 6 Breaking run instance solutions

Event	Vehicle	Order
(a) Emergent solution with advised rules		
$A*$	1	1
A	2	2
C	2	3
B	1	4
E	2	5
G	1	6
D	1	7
F	2	8
(b) Minus pro-active rule 1		
$A*$	1	1
A	2	2
C	2	3
B	1	4
D	1	5
G	2	6
E	2	7
F	1	8

with the A event. As a result, it is termed $A*$ and given a thick dotted border for its delivery location in Fig. 13.

Now, Table 6a and Fig. 13a, plot the behavior of the self-organizing system's response with the four exception rules built up for the recurring set of events from the previous example. As before, a solid arrow path is used for the first agent Ag_1 and a dotted arrow path for the second agent Ag_2.

Author's personal copy

Ag_1 completes the events $A* - B - G - D$ in that order. Note, misbehavior is caused by Ag_1 following pro-active rule 1 after finishing the $A*$ delivery. This is due to Ag_1 realizing it saw the A event appear so it moves to the C pickup location. However, Ag_2 has completed the C event so it moves on to B. After completing B Ag_1 attempts to follow pro-active rule 2 and moves to E before aborting that attempt because of Ag_2 again. Ag_1 also fails to follow pro-active rule 3 because its delays allowed Ag_2 to complete it first.

Ag_2 responds to events $A - C - E - F$. This is a straight-forward path except for after the C event where Ag_2 partially responds to the B event as a result of not encountering Ag_1's infochemicals until almost reaching it.

The total travel cost is ∼128.95 and the extra distance traveled is primarily due to Ag_1 following pro-active rule 1. This rule caused a cascading change in the emergent behavior of agent Ag_2 and resulted in many inefficient choices made by the agents.

However, if pro-active rule 1 is not advised to the system Table 6b and Fig. 13b is the result. Ag_1 now completes $A* - B - D - F$ and Ag_2 $A - C - G - E$. Ag_1 no longer detours to attempt the C delivery, which means it responds to the B delivery quicker, signaling sooner to Ag_2 not to waste the distance traveled to respond to it. Ag_1 is also prevented from following pro-active rule 2 as the result of Ag_2 being closer and starting the E event before the rule is triggered. Ag_1 also now correctly follows pro-active rule 3 to complete the F delivery. The total distance traveled is now ∼89.6.

In summary, this example demonstrated that particular non-recurring events can cause surprisingly unwelcome emergent misbehavior when combined with specific exception rules. This motivates the addition of ELES into the RA-EIA rule derivation process to judge the extent of emergent misbehavior possible as the result of advising a particular prospective rule.

8.4.2 ELES dynamic performance risk experiments

To establish the existence of an exploitation risk due to dynamic change after adaptation this section performs experiments featuring ELES exploratory testing. In this testing the RA-EIA is limited to only **assess**$_{\text{MCS}}$, which only allows the assessment of the variability of risk and not the extent of risk. This leaves the RA-EIA vulnerable to targeted exploitation similar to the EIA. However, the RA-EIA does limit the quantity of rules which has the possibility of reducing the opportunities for exploitation.

There are two configurations for the experiment: in Table 7 there are only recurring events where the adversary can manipulate all the events, and in Table 8 the adversary can only manipulate the two non-recurring events. The second series is more reasonable as it can usually be expected that the generators of recurring events are trusted users of the system.

Similar to the ELES experiments in Sect. 8.3.2, there are twenty experiments for each of these configurations, ten for each of 4 and 6 combinations of recurring events. Of each ten, five are for each of the EIA and RA-EIA. The result of each experiment is an example consisting of a pair of run instances, of which the first is used to adapt the system to a recurring set of events, and the second which is subsequently encountered and solved less efficiently than it would have been before adaptation. See Sect. 5.3 for details of the ELES configuration. Each example is simulated five times by each of the EIA/RA-EIA and averaged to produce a series entry in Tables 7a, b, and 8a, b. Reported is the baseline inefficiency for the second of the pair of run instances along with the level to which the EIA and RA-EIA increased this inefficiency as the result of adaptations from the first run instance.

In the first configuration, Table 7a and b demonstrate that an unlimited adversary can exploit advised exception rules to produce inefficiency. There are examples for each of the

Author's personal copy

Table 7 ELES dynamic performance risk (unlimited)

Test		Performance inefficiency			Exception rules			
Advisor	Series	Baseline (%)	EIA (%)	RA-EIA (%)	Min	Max	EIA	RA-EIA
(a) 4R0N								
EIA	1	116	382	116	4	6	6.0	4.0
	2	86	341	341	4	6	6.0	4.0
	3	70	207	207	4	5	5.0	4.0
	4	78	251	251	2	7	7.0	2.0
	5	60	100	132	3	5	5.0	3.0
RA-EIA	1	32	243	243	4	4	4.0	4.0
	2	37	62	85	2	5	5.0	2.0
	3	117	117	298	2	2	2.0	2.0
	4	24	67	108	2	4	2.8	3.6
	5	41	102	116	3	4	3.2	3.4
(b) 6R0N								
EIA	1	25	49	32	3	8	6.4	3.2
	2	45	57	51	5	7	6.2	5.0
	3	43	124	63	3	7	7.0	4.4
	4	47	164	77	5	8	7.2	5.2
	5	55	121	118	6	8	7.6	6.2
RA-EIA	1	41	153	118	3	7	6.8	4.2
	2	47	90	115	2	8	7.6	2.6
	3	45	98	98	1	7	6.4	2.8
	4	39	76	91	3	9	9.0	4.0
	5	86	118	119	2	8	7.2	3.4

EIA and RA-EIA where the inefficiency of the advised system is drastically larger than the inefficiency of the base system. Despite advising fewer exception rules than the EIA, the RA-EIA is still vulnerable to exploitation. This is evident in the ability of the exploratory search in finding examples for the RA-EIA where the advised exception rules could still be exploited. The average increase in inefficiency for the RA-EIA was from ∼66.1 to ∼189.7 % for $4R0N$ and from ∼47.3 to ∼88.2 % for $6R0N$.

Similarly, Table 8a and b show that despite the adversary being more limited, properly targeted manipulation can still be successful. On average the increase of inefficiency for the RA-EIA was from ∼49.1 to ∼98.7 % for $4R2N$ and from ∼28.3 to ∼63.2 % for $6R2N$.

8.4.3 Managing dynamic performance risk examples

In this section a single example from each of the above experimental series for $6R0N$ and $6R2N$ is chosen. The behavior of the RA-EIA is then examined with and without the use of the **assess**$_{\text{ELES}}$ sub-action, which is designed to avoid advising rules that exceed a chosen threshold of acceptable risk. The first example consists of the run instance pair from the second $6R0N$ experimental series of the RA-EIA (RA-EIA-2), and the other is the third $6R2N$ experimental series of the RA-EIA (RA-EIA-3). In the first example the adversary could manipulate all six events and in the second only the two non-recurring events. It should

Author's personal copy

Table 8 ELES dynamic performance risk (limited)

Test		Performance inefficiency			Exception rules			
Advisor	Series	Baseline (%)	EIA (%)	RA-EIA (%)	Min	Max	EIA	RA-EIA
(a) 4R2N								
EIA	1	15	102	29	1	5	5.0	1.0
	2	29	86	66	2	5	5.0	2.0
	3	56	134	71	2	5	5.0	2.0
	4	36	164	43	1	4	4.0	1.0
	5	79	156	85	3	4	4.0	3.0
RA-EIA	1	56	137	145	3	5	5.0	3.0
	2	24	57	100	2	5	5.0	2.0
	3	72	112	87	1	2	1.2	1.0
	4	46	106	209	3	5	5.0	3.0
	5	78	57	152	1	5	5.0	1.0
(b) 6R2N								
EIA	1	42	104	53	3	7	7.0	3.0
	2	22	79	27	1	7	7.0	1.0
	3	19	92	39	3	6	6.0	3.0
	4	13	47	13	0	7	7.0	0.0
	5	33	96	46	2	5	5.0	2.0
RA-EIA	1	43	68	111	3	6	6.0	3.0
	2	11	29	55	1	10	10.0	1.0
	3	37	23	98	4	7	7.0	4.0
	4	24	43	62	5	7	7.0	5.0
	5	39	46	128	2	7	7.0	2.0

Table 9 Summary comparison of RA-EIA for 6R0N example

	Assess$_{\text{MCS}}$ only		Both	
Distance	ri^{adapt}	ri^{break}	ri^{adapt}	ri^{break}
Optimal	69.94112	41.21320	69.94112	41.21320
Base	95.39699	60.76956	95.39699	60.76956
Advised	78.08327	97.88229	82.32592	82.56857
Inefficiency	ri^{adapt}	ri^{break}	ri^{adapt}	ri^{break}
Base (%)	36	47	36	47
Advised (%)	12	138	18	100
(+/−) (%)	−24	+91	−18	+53

be noted that the second example is the source of the example described in Sect. 8.2 and continued in Sect. 8.4.1.

A summary of the simulation of the $6R0N$ example is given in Table 9. Note, the chosen threshold for **assess**$_{\text{ELES}}$ is $y_{\text{threshold}} = 1.00$. This threshold was chosen to be below the performance inefficiency (1.15) of the example. The goal is that the RA-EIA should allow some adaptation and improve efficiency while managing to identify and avoid adaptations

Author's personal copy

Table 10 Description of 6R0N example derivation

Accumulated	Derived	After ELES	After MCS	Action
(a) Without ELES				
{}	{1,2}	N/A	{1}	Add 1
{1}	{2}	N/A	{2}	Add 2
{1,2}	{3}	N/A	{}	None
{1,2}	{4}	N/A	{}	None
{1,2}	{5}	N/A	{5}	Add 5
{1,2,5}	{3}	N/A	{}	None
{1,2,5}	{4}	N/A	{}	None
{1,2,5}	{6}	N/A	{6}	Add 6
{1,2,5,6}	{3}	N/A	{}	None
{1,2,5,6}	{4}	N/A	{}	None
{1,2,5,6}	{}	N/A	{}	Done
(b) With ELES				
{}	{1,2}	{2}	{2}	Add 2
{2}	{1}	{}	{}	None
{2}	{3}	{3}	{}	None
{2}	{4}	{4}	{}	None
{2}	{5}	{5}	{5}	Add 5
{2,5}	{1}	{}	{}	None
{2,5}	{3}	{3}	{}	None
{2,5}	{4}	{4}	{}	None
{2,5}	{6}	{6}	{}	None
{2,5}	{}	{}	{}	Done

that produced the full extent of emergent misbehavior previously discovered. Reported is the comparison of the RA-EIA with only MCS, and the RA-EIA with both ELES and MCS. The estimated optimal distance traveled necessary to solve each of the run instances that make up the pair is reported along with the distance traveled with and without the exception rule adaptations the RA-EIA advises. These adaptations are created as the result of ri^{adapt} being the recurring set of events. A normalized inefficiency value is given for the distance traveled relative to the estimated optimal in the bottom rows of the chart. As well, the $(+, -)$ row reports the differential between the base and advised inefficiencies, with a negative value indicating an improvement as a result of the advised rules.

Without the **assess**$_{\text{ELES}}$ sub-action the RA-EIA improved the first run instance ri^{adapt} 24 % points as seen in the $(+, -)$, however this resulted in a 91 % point loss when encountering the second run instance ri^{break}. With the **assess**$_{\text{ELES}}$ sub-action the RA-EIA identified and avoided advising a rule resulting in a lower improvement of 18 % points for ri^{adapt}, but successfully reducing the extent of loss to 53 % points for ri^{break}.

The RA-EIA derivation process with and without **assess**$_{\text{ELES}}$ can be contrasted between Table 10a and b. This process reports the current rules accumulated, the derived rules proposed for addition to the accumulated set along with what of those prospective rules remain after being filtered by the ELES and MCS actions. Finally, if a derived rule passed the filtering the rule added is then reported. In Table 10b the RA-EIA consistently identified rule one via ELES whenever it was proposed and eliminated it from consideration. Correspondingly,

Author's personal copy

Table 11 Summary comparison of RA-EIA for 6R2N example

	Assess$_{\text{MCS}}$ only		Both	
Distance	ri^{adapt}	ri^{break}	ri^{adapt}	ri^{break}
Optimal	50.38477	67.11269	50.38477	67.11269
Base	76.18379	91.59800	76.18379	91.59800
Advised	60.87007	128.95338	62.28429	89.59800
Inefficiency	ri^{adapt}	ri^{break}	ri^{adapt}	ri^{break}
Base (%)	51	36	51	36
Advised (%)	21	92	24	34
(+/-) (%)	−30	+56	−27	−2

Table 12 Description of 6R2N example derivation

Accumulated	Derived	After ELES	After MCS	Action
(a) Without ELES				
{}	{1,2}	N/A	{1}	Add 1
{1}	{2}	N/A	{}	None
{1}	{3,4}	N/A	{4}	Add 4
{1,4}	{2}	N/A	{2}	Add 2
{1,2,4}	{3}	N/A	{}	None
{1,2,4}	{5}	N/A	{}	None
{1,2,4}	{6}	N/A	{6}	Add 6
{1,2,4,6}	{3}	N/A	{}	None
{1,2,4,6}	{5}	N/A	{}	None
{1,2,4,6}	{}	N/A	{}	Done
(b) With ELES				
{}	{1,2}	{2}	{}	None
{}	{3,4}	{3,4}	{4}	Add 4
{4}	{1,2}	{2}	{2}	Add 2
{2,4}	{1,3}	{}	{}	None
{2,4}	{5}	{5}	{}	None
{2,4}	{6}	{6}	{6}	Add 6
{2,4,6}	{1,3}	{}	{}	None
{2,4,6}	{5}	{5}	{}	None
{2,4,6}	{}	{}	{}	Done

rule six was subsequently observed to no longer be beneficial without rule one and it was not accumulated. Resultantly, without the **assess**$_{\text{ELES}}$ sub-action the RA-EIA advised the set {1, 2, 5, 6} of rules and advised the set {2, 5} with it.

On the other hand, Table 11 summarizes the simulation of the $6R2N$ example. Note that the chosen threshold for $assess_{\text{ELES}}$ is $y_{\text{threshold}} = 0.75$. This threshold was chosen to be below the performance inefficiency (0.98) of the example. Note, details about ri^{adapt} are available in the example of Sect. 8.2, while details of ri^{break} are available in Sect. 8.4.1.

As before, without $assess_{\text{ELES}}$ the RA-EIA improved the first run instance ri^{adapt} 30 % points, however this resulted in a 56 % point loss when encountering the run instance ri^{break}. With the $assess_{\text{ELES}}$ sub-action the RA-EIA identified and eliminated an exception rule that

Author's personal copy

was previously advised. The result is a slightly lower improvement of 27 % points for ri^{adapt}, but it completely eliminated the efficiency loss for ri^{break}.

The RA-EIA derivation process with and without $assess_{\text{ELES}}$ can be contrasted between Table 12a and b. The RA-EIA consistently identified and eliminated rule one, advising the set {2, 4, 6} of rules when using $assess_{\text{ELES}}$ instead of {1, 2, 4, 6}. This eliminated rule was pro-active rule 1 from the example in Sect. 8.2. The other three rules from the example are the ones that are retained by the RA-EIA.

Overall, the first experiments of this section of the evaluation established that the RA-EIA is still vulnerable to a malicious adversary if it is only using MCS to assess risks. The experiments that followed demonstrated that this risk of exploitation could be assessed and managed pro-actively by the RE-EIA through the use of integrated and automated ELES exploratory testing. It is important to note that in these second experiments the RA-EIA continued to improve efficiency by advising beneficial rules and ignoring those with unacceptable efficiency loss when exploited.

8.5 Discussion of running time

First, the increased complexity of the RA-EIA's rule derivation process must be acknowledged. A consultation with the RA-EIA will take longer than with the EIA. This is due to the EIA's simple, but naive, process of returning the first exception rule derived from the current differences between the emergent and estimated optimal solution to the recurring set of events. The RA-EIA performs a more extensive reflective process during derivation which produces a complete set of exception rules using MCS and/or ELES.

The base self-organizing system's simulation run time is completely separate from the EIA/RA-EIA advisor, therefore there is no change in this run time whether or not either advisor is used. A single PDP run instance for the base self-organizing system is representative of a single day and a run sequence as a number of days. The only major computation for the base EIA is to determine the estimated optimal for a run instance. This is completed for a run instance in the experiment problems in under a minute. This time, even for multiple run instances, is largely irrelevant if the run instance it is associated with represents a day. One consultation of the EIA in the experiments took under a minute.

When only MCS is used by the RA-EIA, rule derivation is accomplished within a matter of a few minutes on a desktop computer for even the largest problem complexity considered in this paper of $8R4N$. This derivation time is based on the sample size required by the MCS and the number of times the rule derivators were reset from the accumulation of a new rule. Reducing the confidence and error of the MCS process reduces this derivation time at the result of losing accuracy. One consultation with the RA-EIA took at most a few minutes and therefore can be exchanged with the EIA without introducing too much additional computation cost. An ELES experiment for the RA-EIA, in Sects. 8.3.2 and 8.4.2, took at most 4 h. This length of time was reached when completing an exploratory search for a complexity of $8R4N$ using a genetic algorithm population of 25 individuals run for 100 generations.

The consultation of ELES during rule derivation is comparable to completing one of the exploratory testing series used to evaluate the system. Each one of these lasted about half of an hour. Therefore, each complete running time for the RA-EIA when using its ELES sub-action in Sect. 8.4.3 to assess risk took a matter of ~44 h on a desktop computer. This is a consequence of using the **assess**$_{ELES}$ action 10 times, in order to take the maximum of the combined results. There is evidence that the length of each run and the number of times

Author's personal copy

assess$_{ELES}$ is called to determine the max can be reduced. However, this type of optimization of the methodology has yet to be explored.

It is clear that the RA-EIA's use of ELES consultation is very unlikely to be completed daily, between every run instance, but rather on a weekly, monthly, or even yearly basis. The RA-EIA completes a process similar to that which a consultant would provide. A consultant is used irregularly, such as once every few months, or even once a year. The RA-EIA's reflective consultation process is completed offline from the advised client system and the changes are assessed to provide long-term risk averse efficiency improvement relevant over the period of time between being consulted. It is important to judge the running time of the RA-EIA with ELES against the stability of the problem being solved to determine if advice would remain valid and if the costs of consultation are exceeded by the efficiency gains over the client system's operation. This is similar to considerations made before making use of any consultation agency's services.

9 Conclusions and future work

Using autonomous agents to form a self-organizing system is a highly desirable solution for dynamic problems requiring scalability and flexibility. However, not all collective behaviors that emerge within such an implementation are beneficial. For many logistical and industrial applications cost efficiency is a necessary behavioral requirement. However, traditionally cost efficiency is a secondary concern when developing self-organizing systems. A consultation agent, such as the EIA, has been shown to provide improvements to efficiency by adapting a self-organizing system through advised exception rules. The derivation of these exception rules is based on the assumption that there are recurring characteristics in the encountered problem. In order for the operators of such an advised self-organizing system to trust a proposed real-world implementation they must be assured that the risks associated with costly emergent misbehavior are mitigated through preemptive risk assessment and management. To accomplish this the RA-EIA uses offline reflection, via Monte Carlo Simulation and Evolutionary Learning of Event Sequences, during the derivation and advisement of exception rules. The result is that the advised client system is able to independently operate over a longer period of time while maintaining efficiency, flexibility, and autonomy.

The completed evaluation supports the claimed capabilities of the RA-EIA. First, the EIA and RA-EIA's comparative ability to manage the expected risks from derived exception rules was examined. The EIA demonstrated that lacking reflection its rule derivation process often resulted in an excessive count of rules and, more importantly, rules that could decrease performance. In contrast, the RA-EIA's use of Monte Carlo Simulation allowed it to advise a minimal, yet effective, set of exception rules. The use of reflection enabled the RA-EIA to achieve greater efficiency gains for the experiments with randomly generated events and less risk of performance efficiency loss for the biased experiments where it was challenged by extreme examples discovered through the use of Evolutionary Learning of Event Sequences. Finally, when only using Monte Carlo Simulation the RA-EIA, just like the EIA, was vulnerable having advised rules be exploited. However, the addition of risk assessment through an internalized usage of Evolutionary Learning of Event Sequences allowed the RA-EIA to be able to assess and manage the extent of this exploitation and limit rules to those that are difficult for a malicious adversary to take advantage of.

There are a number of areas for future work. Many different variants of exception rules have not been assessed. Similarly, other performance goals, such as security and robustness against failure remain unexplored. Additionally, there are possible applications of the RA-

Author's personal copy

EIA concept in which it may be desirable to integrate information newly communicated by agents during the rule derivation process. This circumstance is more likely within problems that do not decompose into day like periods with obvious downtime in which the RA-EIA rule derivation process best integrates.

Evolutionary Learning of Event Sequences is an automated form of exploratory testing. Its development was initially proposed to automatically assess performance and establish the danger represented by self-adaptation. Test driven development may be possible by embedding it within the feedback cycle of development to assess the extent of misbehavior, examining the produced examples for inefficiencies, determining adjustments and improvements to counter the examples, and iterating the cycle to reassess misbehavior.

References

1. Akour, M., Jaidev, A., & King, T.M. (2011). Towards Change Propagating Test Models in Autonomic and Adaptive Systems. In *Proceedings of the International Conference on Engineering of Computer-Based Systems, ECBS '11* (pp. 89–96). IEEE Computer Society.
2. Barbucha, D., & Jedrzejowicz, P. (2009). Agent-based approach to the dynamic vehicle routing problem. In *Proceedings of the International Conference on Practical Applications of Agents and Multi-Agent Systems, PAAMS' 09* (pp. 169–178). Berlin: Springer.
3. Berbeglia, G., Cordeau, J. F., Gribkovskaia, I., & Laporte, G. (2007). Static pickup and delivery problems: A classiffication scheme and survey. *TOP*, *15*(1), 1–31.
4. Coelho, D. K., Roisenberg, M., de Freitas Filho, P. J., & Jacinto, C. M. C. (2005). Risk assessment of drilling and completion operations in petroleum wells using a monte carlo and a neural network approach. In *Proceedings of the Winter Simulation Conference, WSC '05* (pp. 1892–1897). Winter Simulation Conference.
5. Davidsson, P., Persson, J. A., & Holmgren, J. (2007). On the integration of agent-based and mathematical optimization techniques. In *Proceedings of the International Symposium on Agent and Multi-Agent Systems, KES-AMSTA '07* (pp. 1–10). Berlin: Springer.
6. De Wolf, T., & Holvoet, T. (2005). Emergence versus self-organisation: Different concepts but promising when combined. In S. A. Brückner, G. D. M. Serugendo, A. Karageorgos, & R. Nagpal (Eds.), *Engineering Self-Organising Systems* (Vol. 3464, pp. 1–15)., Lecture Notes in Computer Science Berlin: Springer.
7. De Wolf, T., & Holvoet, T. (2007). A taxonomy for self-* properties in decentralised autonomic computing. In M. Parashar & S. Hariri (Eds.), *Autonomic Computing: Concepts, Infrastructure, and Applications* (pp. 101–120). Boca Raton: CRC Press.
8. Di Stefano, A., Pappalardo, G., Santoro, C., & Tramontana, E. (2007). A framework for the design and automated implementation of communication aspects in multi-agent systems. *Journal of Network and Computer Applications*, *30*, 1136–1152.
9. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., et al. (2006). A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, *1*(2), 223–259.
10. Dorer, K., & Calisti, M. (2005). An adaptive solution to dynamic transport optimization. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS '05* (pp. 45–51). ACM Press.
11. Ferber, J., & Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the International Conference on Multi Agent Systems, ICMAS '98* (pp. 128–135). IEEE Computer Society.
12. Flanagan, T., Thornton, C., & Denzinger, J. (2009). Testing harbour patrol and interception policies using particle-swarm-based learning of cooperative behavior. In *Proceedings of the Computational Intelligence for Security and Defense Applications, CISDA '09* (pp. 1–8).
13. Hartigan, J. A. (1975). *Clustering Algorithms*. New York: Wiley.
14. Horn, P. (2001). *Autonomic Computing: IBM's Perspective on the State of Information Technology*. Tech. rep., IBM.
15. Huang, G., Liu, T., Hong, M., Zheng, Z., Liu, Z., & Fan, G. (2004). Towards Autonomic Computing Middleware via Reflection. *Computer Software and Applications Conference, Annual International*, *1*, 135–140.
16. Hudson, J. (2011). *Risk assessment and management for efficient self-adapting self-organizing emergent multi-agent systemss*. Master's thesis, University of Calgary.

Author's personal copy

17. Hudson, J., Denzinger, J., Kasinger, H., & Bauer, B. (2009). *Testing Self-Organizing Emergent Systems by Learning of Event Sequences*. Technical Report 2009–949-28, Department of Computer Science, University of Calgary.
18. Hudson, J., Denzinger, J., Kasinger, H., & Bauer, B. (2010). Efficiency testing of self-adapting systems by learning of event sequences. In *Proceedings of the International Conference on Adaptive and Self-adaptive Systems and Applications, ADAPTIVE '10* (pp. 200–205).
19. Hudson, J., Denzinger, J., Kasinger, H., & Bauer, B. (2011). Dependable risk-aware efficiency improvement for self-organizing emergent systems. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems, SASO '11* (pp. 11–20).
20. IBM (2006). Autonomic Computing Whitepaper: An Architectural Blueprint for Autonomic Computing. Technical Report.
21. IBM (2008). *The GMA 2008 logistics survey: Improving efficiency in the face of mounting logistics costs*. http://www-935.ibm.com/services/us/gbs/bus/pdf/gbe03063-usen-gma08.
22. Jain, R. (1991). *The Art of Computer Systems Performance Analysis*. New York: Wiley-Interscience.
23. Jelasity, M., Babaoglu, O., & Laddaga, R. (2006). Self-management through self-organization. *IEEE Intelligent Systems*, *21*(2), 8–9.
24. Kaner, C., Falk, J., & Nguyen, H. Q. (1993). *Testing Computer Software*. Boston: International Thomas Computer Press.
25. Kasinger, H. (2010). *Design and Operation of Efficient Self-Organizing Systems*. Ph.D. thesis, Universität Augsburg.
26. Kasinger, H., Bauer, B., & Denzinger, J. (2008). The meaning of semiochemicals to the design of self-organizing systems. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems, SASO '08* (pp. 139–148). IEEE Computer Society.
27. Kasinger, H., Bauer, B., & Denzinger, J. (2009). Design pattern for self-organizing emergent systems based on digital infochemicals. In *Proceedings of the International Conference and Workshops on Engineering of Autonomic and Autonomous Systems, EASe '09* (pp. 45–55). IEEE Computer Society.
28. Kasinger, H., Bauer, B., Denzinger, J., & Holvoet, T. (2010). Adapting environment-mediated self-organizing emergent systems by exception rules. In *Proceedings of the International Workshop on Self-Organizing Architectures, SOAR '10* (pp. 35–42). ACM Press.
29. Kasinger, H., Denzinger, J., & Bauer, B. (2008). Digital semiochemical coordination. *Communications of SIWN*, *4*, 133–139.
30. Kasinger, H., Denzinger, J., & Bauer, B. (2009). Decentralized coordination of homogeneous and heterogeneous agents by digital infochemicals. In *Proceedings of the Symposium on Applied Computing, SAC '09* (pp. 1223–1224). ACM Press.
31. Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*, *36*(1), 41–50.
32. King, T. M., Allen, A. A., Wu, Y., Clarke, P. J., & Ramirez, A. E. (2011). A comparative case study on the engineering of self-testable autonomic software. In *Proceedings of the International Conference and Workshops on Engineering of Autonomic and Autonomous Systems, EASe '11* (pp. 59–68). IEEE Computer Society.
33. Maes, P., & Nardi, D. (Eds.). (1988). *Meta-Level Architectures and Reflection*. New York: Elsevier Science Inc.
34. Mahr, T., Srour, J., de Weerdt, M. M., & Zuidwijk, R. (2010). Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Transportation Research: Part C*, *18*(1), 99–119.
35. McCabe, B. (2003). Monte Carlo simulation for schedule risks. In *Proceedings of the Winter Simulation Conference, WSC '03* (pp. 1561–1565). Winter Simulation Conference.
36. Mogul, J. C. (2006). Emergent (mis)behavior vs. complex software systems. *ACM SIGOPS Operating Systems Review*, *40*(4), 293–304.
37. Nakrani, S., & Tovey, C. (2004). On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, *12*, 223–240.
38. Nguyen, C., Miles, S., Perini, A., Tonella, P., Harman, M., & Luck, M. (2012). Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, *25*, 260–283.
39. NIST/SEMATECH (2011). *e-Handbook of Statistical Methods*. http://www.itl.nist.gov/div898/handbook/.
40. Ogata, K. (2009). *Modern Control Engineering*. Boston: Prentice Hall.
41. Pěchouček, M., Štepánková, O., Mařík, V., & Bárta, J. (2003). Abstract architecture for meta-reasoning in multi-agent systems. In *Proceedings of the Central and Eastern European conference on Multi-agent systems, CEEMAS'03* (pp. 84–99). Springer-Verlag.

Author's personal copy

42. Richter, U., Mnif, M., Branke, J., Müller-Schloer, C., & Schmeck, H. (2006). Towards a generic observer/controller architecture for Organic Computing. In *Proceedings of Informatik, Informatik '06* (pp. 112–119).
43. Sailing, K. B., & Leleur, S. (2006). Assessment of transport infrastructure projects by the use of Monte Carlo simulation: The CBA-DK model. In *Proceedings of the Winter Simulation Conference, WSC '06* (pp. 1537–1544). Winter Simulation Conference.
44. Sailing, K. B., & Leleur, S. (2007). Appraisal of airport alternatives in greenland by the use of risk analysis and monte carlo simulation. In *Proceedings of the Winter Simulation Conference, WSC '07* (pp. 1986–1993). Winter Simulation Conference.
45. Schmitt, A., & Singh, M. (2009). Quantifying supply chain disruption risk using Monte Carlo and discrete-event simulation. In *Proceedings of the Winter Simulation Conference, WSC '09*, (pp. 1237–1248). Winter Simulation Conference.
46. Schumann, R., Lattner, A. D., & Timm, I. J. (2008). Management-by-exception—a modern approach to managing self-organizing systems. *Communications of SIWN*, *4*, 168–172.
47. Serugendo, G. D. M., Gleizes, M. P., & Karageorgos, A. (2005). Self-organization in multi-agent systems. *The Knowledge Engineering Review*, *20*(2), 165–189.
48. Shaw, M. (1995). Beyond objects: A software design paradigm based on process control. *ACM Software Engineering Notes*, *20*(1), 27–38.
49. Steghöfer, J. P. (2008). *Improving efficiency and coordination in self-organizing emergent systems*. Master's thesis, Universität Augsburg.
50. Steghöfer, J. P., Denzinger, J., Kasinger, H., & Bauer, B. (2010). Improving the efficiency of self-organizing emergent systems by an advisor. In *Proceedings of the International Conference and Workshops on Engineering of Autonomic and Autonomous Systems, EASe '10* (pp. 63–72). IEEE Computer Society.
51. Steiner, T. (2010). *Improving Self-organizing Self-adapting Multi-Agent Systems using Predictive Exception Rules*. Master's thesis, Universität Augsburg.
52. Steiner, T., Denzinger, J., Kasinger, H., & Bauer, B. (2011). Pro-active advice to improve the efficiency of self-organizing emergent system. In *Proceedings of the International Conference and Workshops on Engineering of Autonomic and Autonomous Systems, EASe '11* (pp. 97–106). IEEE Computer Society.
53. Sterritt, R., Garrity, G., Hanna, E., & O'Hagan, P. (2006). Survivable security systems through autonomicity. In M. Hinchey, P. Rago, J. Rash, C. Rouff, R. Sterritt, & W. Truszkowski (Eds.), *Innovative Concepts for Autonomic and Agent-Based Systems* (Vol. 3825, pp. 379–389)., Lecture Notes in Computer Science Berlin: Springer.
54. Toth, P., & Vigo, D. (2002). *The Vehicle Routing Problem*. Philadelphia, PA: Society for Industrial and Applied Mathematics.
55. Truszkowski, W. F., Hinchey, M. G., Rash, J. L., & Rouff, C. A. (2006). Autonomous and autonomic systems: A paradigm for future space exploration missions. *IEEE Transactions on Systems, Man, and Cybernetics—Part C*, *36*(3), 279–291.
56. Weyns, D., Boucké, N., & Holvoet, T. (2008). A field-based versus a protocol-based approach for adaptive task assignment. *Autonomous Agents and Multi-Agent Systems*, *17*(2), 288–319.
57. Weyns, D., & Georgeff, M. (2010). Self-adaptation using multi-agent systems. *IEEE Software*, *27*(1), 86–91.
58. Weyns, D., Haesevoets, R., Van Eylen, B., Helleboogh, A., Holvoet, T., & Joosen, W. (2008). Endogenous versus exogenous self-management. In *Proceedings of the International Workshop on Software Engineering for Adaptive and Self-managing Systems, SEAMS '08* (pp. 41–48). ACM Press.
59. Wooldridge, M. J., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *10*(2), 115–152.
60. Wu, D., Song, H., Li, M., Cai, C., & Li, J. (2010). Modeling risk factors dependence using Copula method for assessing software schedule risk. In *Proceedings of the Software Engineering and Data Mining Conference, SEDM '10* (pp. 571–574).