Procedia Computer Science

Volume 29, 2014, Pages 1057–1067

ICCS 2014. 14th International Conference on Computational Science

Agent-based Adaptation System for Service-Oriented Architectures Using Supervised Learning

Bartłomiej Śnieżyński

AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecomunications
Department of Computer Science, Krakow, Poland
Bartlomiej.Sniezynski@agh.edu.pl

Abstract

In this paper we propose an agent-based system for Service-Oriented Architecture self-adaptation. Services are supervised by autonomous agents which are responsible for deciding which service should be chosen for interoperation. Agents learn the choice strategy autonomously using supervised learning. In experiments we show that supervised learning (Naïve Bayes, C4.5 and Ripper) allows to achieve much better efficiency than simple strategies such as random choice or round robin. What is also important, supervised learning generates a knowledge in a readable form, which may be analyzed by experts.

Keywords:

1 Introduction

Service-Oriented Architecture (SOA) is one of the common techniques used to build large scale systems, which gives very good results [35]. However, there are problems with huge number of parameters that should be configured to achieve assumed Quality of Service (QoS). Therefore, there are many works on automatic adaptation of SOA [6, 29].

Large scale systems are usually distributed, which makes of them a natural environment for agent-based solutions. Such environment is complex and changing. As a consequence, it is very difficult to design to design all system details a priori. To overcome this problem one can apply a learning algorithm which allows to adapt agents to the environment.

There are many machine learning algorithms developed so far. However, in multi-agent systems most applications use reinforcement learning [19, 22, 32]. The problem is that in a complex environment (where state-space is large) reinforcement learning needs time to reach satisfactory performance and the knowledge learned is very difficult to analyze. These problems

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

suggest to search for other solutions, like supervised learning. It appears that it can be also applied in multi-agent systems and gives results faster and in a human-readable form [25, 26].

Goal of this research is to develop agent-based system for SOA self-adaptation with the following two main assumptions: services are supervised by autonomous agents which are responsible for deciding which service should be chosen for interoperation; agents learn the choice strategy autonomously using supervised learning.

As a result, we make the following contributions to the state of the art: we propose agent-based system for SOA adaptation; we show that methods other than reinforcement learning can be used by agents to generate its strategy; we show that knowledge learned using this technique has readable form.

In the following section we overview related research. Next, the agent architecture and learning mechanism is described. It is followed by the presentation of experimental results. Finally, conclusions and the further work are outlined.

2 Related Research

Let us start with Adaptation in SOA. According to McKinley et al. [18], software adaptation is an adjustment of the system structure or parameters in response to changes in the execution environment. Berry et al. [4] define Dynamic Adaptive System (DAS), which recognizes changes in the environment and is capable of changing its behavior to adapt to these changes. System is self-adaptive if adaptation is based on the feedback from the system or the context (surrounding environment) [21].

Moving to practical solutions, IBM proposed the Architectural Blueprint [8]. Sensors gather data about the managed resource. These data are monitored and analyzed. Next plan is prepared and executed using effectors. All the phases are based on a common knowledge. Therfore this loop is called MAPE-K (Monitore, Analyse, Plan, Execute - Knowledge). Similar approach is presented by Salehie and Tahvildari [21]. Another practical example is StarMX. It is a framework for Java enterprise environments [2]. It provides support for dynamic adaptation and development of self-managing applications. StarMX can be integrated with various policy/rule engines to enable self-management capabilities.

Vuković investigates context aware applications, which are able to adapt to changes in the environment [33]. Planning technologies are used to prepare the sequence of services execution for given parameters (resources available, time constraints, and user location). The system developed is able to handle composition failures.

In [36] concept of an Adaptive SOA Solution Stack (AS3) is presented. The pattern provides components constituting an adaptation loop. In continuation of this research machine learning algorithms are added. Results are presented in [24], where clustering algorithms are used to group similar system states into clusters and reinforcement learning is used to learn the adaptation strategy: what action should be executed in a given state to achieve better Quality of Service (QoS).

Agent-based technologies are considered as a tool that can be very useful in SOA applications [12]. Good example is work done by Piunti et al. [20]. General-purpose programming model based on BDI agent technologies [] is proposed and applied for developing SOA/WS applications.

In agent domain adaptation and learning are common research topics [19, 22, 32]. The most common learning strategy applied in agent-based systems is reinforcement learning [28]. This learning strategy allows to generate a strategy for an agent in a domain, in which the environment provides some feedback after the agent has acted. Feedback takes the form of a real number representing reward, which depends on the quality of the action executed by the

agent in a given situation. What is important, the reward can be delayed. The goal of the learning is to maximize estimated reward. A typical example of domain used in such works is a Predator-Prey environment [30], where predator agents use reinforcement learning to learn a strategy minimizing time to catch a prey.

There is only a small number of works known to the author on supervised learning in multi-agent systems. Typical examples are discussed below.

Rule induction is used in a multi-agent solutions for vehicle routing problems [10]. However, in this work learning is done off-line. First, rules are generated by the AQ algorithm (the same as used in this work) from traffic data. Next, agents use these rules to predict traffic. Extension of this work is [27]. Agents use hybrid learning algorithm. Rule induction is used to decrease the size of the search space for reinforcement learning.

Airiau et al. [1, 23] adds learning capabilities to the BDI model. Decision tree learning is used to support plan applicability testing. Each plan has its own decision tree to test if it may be used in a given context. As a result, plans may be modified by providing additional conditions limiting its applicability. Knowledge learned has an indirect impact on the agent strategy because it has influence on probability of choosing plans for execution.

There are several works in which Inductive Logic Programming (ILP) is applied. A good background paper considering machine learning and especially ILP for multi-agent systems is [15]. Supervised learning (the subject of this paper) can be considered as a special case of ILP, where simple logic program defining one predicate only is learned. ILP shows its advantage over classical rule induction in complex domains, whereas most multi-agent applications are relatively simple (see conclusions of [15]). Therefore, supervised learning seems to be enough in most cases.

Comparison of various learning strategies in the same domain can be found in [26]. The paper presents a Farmer-Pest domain, which is especially designed for learning agent benchmarking. Every agent controls a farmer that supervises multiple fields. Several types of pests can appear on each field, and the farmer should execute an action which is appropriate for the pest type. Each pest is described by a set of attributes (e.g. number of legs, color) which are visible to the agent, while the pest type is hidden. Therefore, agents have to learn how to assign an action to the observed conditions represented by the attributes. In the paper agents using reinforcement learning (SARSA) are compared with ones using supervised learning algorithms (Naïve Bayes, C4.5 and RIPPER).

In [25] rule induction is used to generate agent strategy in the Predator-Prey domain and in Fish-Banks game. This type of learning is compared with reinforcement learning. Some aspects of learned knowledge exchange are also investigated. Similar approach may be applied for SOA adaptation. It is described below.

3 Agent-based SOA Adaptation System

The general idea of the system is that service may be accompanied by an agent, which is responsible for choosing other services to process given request. The agent is learning how to chose services using supervised learning. Below agent model for SOA adaptation is described and learning agent architecture is presented.

3.1 Agent Model for SOA adaptation

We assume that the system consists of services $S_1, , S_n$. Every service S_i may interoperate with other services by executing actions provided by them. In a case when some type of action is

implemented by several services $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$, it is possible to use some learning method to build a choice strategy based on the current conditions. It allows for adaptation to specific situation.

In agent-based solution we assume that every service S_i is or cooperates with the *SOA Adaptation Agent* ag_i, which observes state of S_i and its neighborhood (e.g. states of cooperating services) and gathers historical data (experience, training data) about interoperation appearing so far. This experience has a form of the sequence of examples. Every example consists of the chosen service identifier and call results (was the action successful, how much time did the calculations take, what was the quality of the result). It can also consists of other parameters describing e.g.:

- call arguments (e.g. input data size, values of key parameters) ;
- service S_i state (e.g. amount of the free memory, CPU usage, queue length, number of logged users);
- states of the other services, which may interoperate with S_i or are on the same machines $(S_{i_1}, S_{i_2}, \ldots, S_{i_k})$;
- date, hour, day of the week, is it a holiday, etc.

All these parameters can represent current values, average from last dt seconds, several moving averages, etc. Agent actions correspond to service choice. Using experience and machine learning algorithm, agent is able to generate strategy of choosing the service for a given task.

3.2 Agent Architecture

In this section we present the learning agent architecture, which is applied in this research. It is presented in Fig. 1. The agent consists of four main modules:

Processing Module is responsible for basic agent activities, storing training data, executing learning process, and using learned knowledge;

Learning Module is responsible for execution of learning algorithm and giving answers for problems with use of learned knowledge;

Training Data is a storage for examples (experience) used for learning;

Generated Knowledge is a storage for learned knowledge.

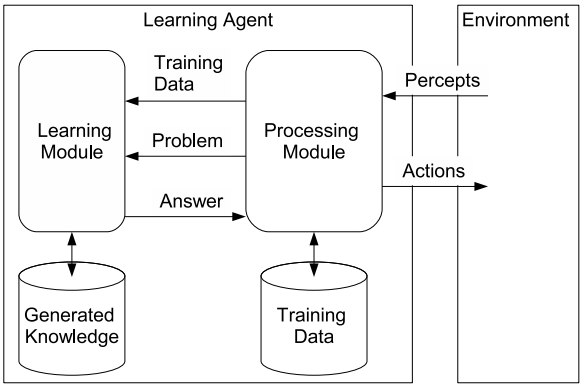

Figure 1: Learning agent architecture

```
begin
    Generated Knowledge:= ∅;
    Training Data:= ∅;
    while agent is alive do
        if Generated Knowledge = ∅ then
            a:= random action
        end
        else
            Problem := description of the current (observed) state;
            a:= action determined for Problem by the classifier stored in Generated
            Knowledge
        end
        execute a;
        if results of the action are interesting (e.g. response was quick enough) then
            store example in the Training Data
        end
        if it is learning time (e.g. every 100 steps) then
            learn from Training Data;
            store knowledge in Generated Knowledge;
        end
    end
end
```

Figure 2: Algorithm of the learning agent

These components interact in the following way. *Processing Module* receives *Percepts* from the environment (parameters listed in subsection 3.1). It may process them and execute *Actions*. If during processing learned knowledge is needed, it formulates a *Problem* and sends it to the *Learning Module*, which generates an *Answer* for the *Problem* using *Generated Knowledge*. *Processing Module* decides also what data should be stored in the *Training Data* storage. When necessary (e.g. periodically, or when *Training Data* contains many new examples) it calls the *Learning Module* to execute the learning algorithm to generate new knowledge from *Training Data*. The learned knowledge is stored in the *Generated Knowledge* base. Algorithm of the agent is presented in Fig. 2.

The form of the knowledge stored in the *Generated Knowledge* depends on the utilized learning algorithm used. It may have an explicit form, like rules or decision tree in the case of supervised learning, which is the case in this research. It can be also stored in a low-level form, like parameters representing action value function or neural network approximator of such a function if reinforcement learning is applied.

Learning module may be defined as a four-tuple: (*Learning Algorithm*, *Training Data*, *Problem*, *Answer*). Characteristics of the training data, the problem and the answer depend on the learning strategy used in the learning module.

3.3 Supervised Strategy Learning

Supervised learning allows to generate a hypothesis $h : X \to C$ which is an approximation of a function $f : X \to C$ which assigns labels from the set C to objects from set X. To generate knowledge a supervised learning algorithm needs training data in a form of labeled

examples which consist of pairs of f arguments and values. Let us assume that elements of X are described by set of attributes $A = (a_1, a_2, \ldots, a_n)$, where $a_i : X \to D_i$. Hence, instead of example x its description $x^A = (a_1(x), a_2(x), \ldots, a_n(x))$ is used. If size of C is small, like in this research, the learning is called classification, C is set of classes, and h is called classifier.

In our case, supervised learning module gets a *Training data*, which is a set of examples describing history of service invocations. Examples consist of attributes describing current time, state of the service invoked and action representing identifier of the service (k). Example label represents quality of the interoperation. In our case it represents response time, but may also aggregate precision of the result, its cost or QoS in general. As a consequence, *Generated Knowledge*, which is obtained from this data, is a classifier which can be used to assign class representing predicted quality to the considered service. The *Problem* is a set of attributes describing current time and service candidate, and the *Answer* is prediction of quality of this service. Therefore, *Processing Module* should classify all $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ services and choose a service which gets highest certainty of good quality.

In experiments we used the following supervised learning algorithms: Naïve Bayes classifier, decision tree learning C4.5, and decision rule induction Ripper.

3.4 Environment Exploration Strategy

In reinforcement learning various techniques are used to prevent from getting into a local optimum [14]. The idea is to explore the solution space better by choosing not optimal actions from time to time (e.g. random or not performed in a given state yet). We decided to use similar technique for supervised learning. 10% actions are random which corresponds to the ϵ-greedy strategy.

3.5 Implementation

To perform experiments, specialized software with service-oriented architecture was implemented [5]. It is written in Java and uses Apache Camel integration platform [13] to apply Enterprise Integration Patterns (EIP) [11]. Apache CXF platform is used to build services [3]. *Serwer Module* is the core part of the system. It provides services, which call external services to perform their tasks. Performance of external services is controlled by *External Services Module* to simulate changing characteristics: average response time, CPU and memory usage. Time is divided into periods with fixed length and these characterisctics are changed periodically. *Load Generator Module* simulates users requesting the *Serwer Module* and allow to test the system performance. Learning algorithms are provided by Weka package [34].

4 Experiments

Using software developed, experiments testing adaptation were performed. System *effectiveness* is defined as the system average response time to user request, which corresponds to assumed QoS.

4.1 Setup

There are three services considered in experiments: S_0, S_1, S_2. SOA Adaptation Agent is responsible for S_0 and observes results of its interoperation with external services S_1, S_2. These services provide information about CPU and memory usage of their hosts, which are described

by *cpu* and *mem* attributes, respectively. These values represent context of the call and are returned together with the response to the service call. As mentioned, characteristics of external services change periodically. Period length is dt. Examples stored in the *Training Data* are described by four attributes: (t, cpu_k, mem_k, a_k), where t is a time from the beginning of the current period, mem_k and cpu_k are S_k parameters, a_k is an action of calling the service S_k (one which was invoked in the given context). Category c of the given example is *good* if S_k was a good choice because execution time was below the average (calculated from four last periods) or *bad* in the other case.

Learning algorithm is executed after finishing every period. *Training Data* cover last four periods to eliminate outdated records. When learning agent chooses the action using knowledge learned, its *Processing module* asks the *Learning module* for the categories of all the possible actions. *Problem* consists of the current context description using the same attributes and service identifier: (t, cpu_k, mem_k, a_k). Action a getting highest certainty of *good* category is chosen. To achieve mentioned exploration, chosen action may be replaced by the other action with $\varepsilon = 10\%$ probability.

In experiments three supervised learning algorithms (implemented in Weka) were applied: C4.5 (J48), Ripper (JRip) and Naïve Bayes. Results achieved by the learning agents are compared with two simple strategies: random algorithm and Round Robin.

Experiment 1 It is assumed that S_1 and S_2 services work quickly and next slowly, alternately. The performance changes in the middle of the period in even periods, and every $\frac{1}{3}dt$ in odd periods. CPU and memory usage changes together with the speed. Its percentage value is randomized using normal distribution with $\sigma^2 = 0.1$ and μ_{mem} =10% and μ_{cpu} =5% when the service is fast, and μ_{mem} =35% and μ_{cpu} =25% when the service is slow. Fig. 3-(1) shows how system efficiency changes in the course of time.

Experiment 2 In this setting it is assumed that during three initial periods both services S_1 and S_2 services work quickly and slowly alternately and service performance changes always in the middle of the period. After fourth period environment characteristics is changed. S_1 service is always slow and S_2 is always fast since then. As above, CPU and memory usage of the machine hosting each service is correlated with its response time. Fig. 3-(2) shows how system efficiency changes in the course of time.

Experiment 3 This experiment assumes that the response time is low only, when both CPU and memory usage are high. In other cases services are fast. Parameters *mem* and *cpu* change periodically but changes have moved phase and different period for S_1 and S_2, which make the time-related pattern more complicated. Change of efficiency is presented in Fig. 3-(3).

4.2 Discussion of the Results

All learning algorithms allowed agents to achieve better results than simple service choice algorithms. One period is enough to collect the experience reach enough to generate efficient knowledge. Statistical tests (t-Student) confirm that the differences are significant at $p < 0.05$.

In the second experiment one may observe small decrease of efficiency for C4.5 and Naïve Bayesafter the characteristics change. These algorithms need to collect training examples in the new conditions. After one or two periods efficiency returns to the previous value.

In the third experiment application of simple service choice algorithms leads to efficiency fluctuations. The reason is that every second period for most of the time both services are

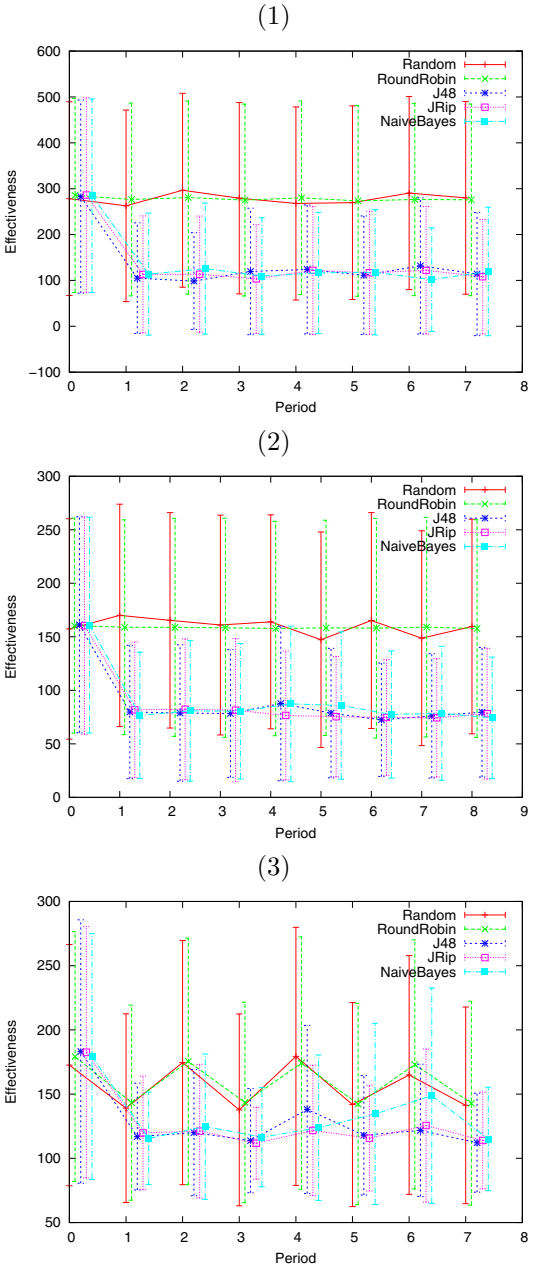

Figure 3: System efficiency changes in the course of time for experiments 1–3 (vertical axis represents mean user waiting time, horisontal axis represents subsequent periods)

quick, and every second period they are slow. In this configuration one can observe more serious decrease of efficiency after the characteristics change. It is especially large when Naïve Bayeslearning algorithm is applied. The reason is that attributes *mem cpu* are not conditionally independent, which makes this case hard for this classifier. What is more, during learning after the fifth period, there are no examples from the first period (with many negative examples) in

```
                (a)                                    (b)
(cpu >= 0.33) and (mem >= 0.4) => class=bad     cpu <= 0.07 :  class=good
                               => class=good    cpu >  0.07
                                                |     mem <= 0.12 :  class=good
                                                |     mem >  0.12 :  class=bad
```

Figure 4: Examples of knolwedge learned by agents in the third experiment in a form of decision rules generated using Ripper algorithm (a) and decision tree generated by C4.5 algorithm (b)

the training data. This makes the data is of lower quality. When training data are filled with more examples, in the last period, Naïve Bayesachieves good results again. However, it shows that in the case of such dependency between attributes Naïve Bayesshould not be applied for learning.

Examples of knowledge learned by agents in the third experiment are presented in Fig. 4. Ripper algorithm built two rules. Premise of the firs one is a conjunction of tests checking if usage level of both resources is high. If so, such a service should not be chosen. The second rule is used if premise of the first one is not fulfiled, which means that if usage of the one of the resources is low then such a service is good.

Decision tree built by C4.5 algorithm checks in the root CPU usage. If it is low, the service is good. In the other case memory usage is analyzed. If it is low, than service is good, in the other case it is bad.

5 Conclusion and Further Research

Results show that supervised learning may be used by agents to generate strategy for service oriented architecture adaptation in the changing conditions. Agents learn autonomously. The learning process allows to achieve better QoS than simple strategies such as random choice or round robin. Because of symbolic representation, knowledge learned may be analyzed and edited by human experts. It also allows to discover patterns not known a priori which can be used by experts to understand distributed system behavior and improve its configuration.

Supervised learning do not suffer from exponential explosion of the state space [9] which is a problem in reinforcement learning [16]. Therefore it can be applied without modifications in environments, where decision about the service choice is based on many parameters [26].

Additional advantage of supervised learning is that the learning process does not need a tuning of several parameters (in the contrast to reinforcement learning). It allows to configure the system easily.

In the future research more complex distributed system configurations will be examined. Testing of communication between agents is also planned. Similar agent-based architecture using supervised learning will be also used in other applications like resource allocation [7], transport systems [17] and Focused Web Crawling [31]. Good application will be also a system for intelligence data analysis that is being developed for the Polish Government Protection Bureau, because of assumed number of heterogeneous services.

Acknowledgments The research reported in the paper was supported by the grant "Information management and decision support system for the Government Protection Bureau" (No. DOBR-BIO4/060/13423/2013) from the Polish National Centre for Research and Development.

References

[1] Stéphane Airiau, Lin Padham, Sebastian Sardina, and Sandip Sen. Incorporating learning in bdi agents. In *Proceedings of the ALAMAS+ALAg Workshop*, May 2008.

[2] Reza Asadollahi, Mazeiar Salehie, and Ladan Tahvildari. Starmx: A framework for developing self-managing java-based systems. *Software Engineering for Adaptive and Self-Managing Systems, International Workshop on*, 0:58–67, 2009.

[3] Naveen Balani and Rajeev Hathi. *Apache CXF Web Service Development*. Packt Publishing, 2009.

[4] Daniel M. Berry, Betty H. C. Cheng, and Ji Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *In 11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ*, 2005.

[5] Micha Bidecki, Marcin Chelmecki, Mateusz Kramarczyk, and Bartlomiej Sniezynski (supervisor). System konfiguracji i analizy serwisow SOA. Engineer Project, AGH, Krakow, 2013.

[6] C. Cappiello, K. Kritikos, A. Metzger, M. Parkin, B. Pernici, P. Plebani, and M. Treiber. A quality model for service monitoring and adaptation. In *Workshop on Monitoring, Adaptation and Beyond (MONA+) at the ServiceWave 2008 Conference*, pages 29–42. Springer, 2008.

[7] Krzysztof Cetnarowicz and Rafal Drezewski. Maintaining functional integrity in multi-agent systems for resource allocation. *Computing and Informatics*, 29(6):947–973, 2010.

[8] IBM Corp. An architectural blueprint for autonomic computing. Technical report, 2005.

[9] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. pages 144–151. Morgan Kaufmann, 1998.

[10] Jan D. Gehrke and Janusz Wojtusiak. Traffic prediction for agent route planning. In Marian Bubak, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *ICCS (3)*, volume 5103 of *Lecture Notes in Computer Science*, pages 692–701. Springer, 2008.

[11] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[12] M. N. Huhns, M. P. Singh, M. Burstein, K. Decker, E. Durfee, T. Finin, L. Gasser, H. Goradia, N. Jennings, K. Lakkaraju, H. Nakashima, V. Parunak, J. S. Rosenschein, A. Ruvinsky, G. Sukthankar, S. Swarup, K. Sycara, M. Tambe, T. Wagner, and L. Zavala. Research directions for service-oriented multiagent systems. *Internet Computing, IEEE*, 9(6):65–70, 2005.

[13] Claus Ibsen and Jonathan Anstey. *Camel in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[15] Dimitar Kazakov and Daniel Kudenko. Machine learning and inductive logic programming for multi-agent systems. In *Multi-Agent Systems and Applications*, pages 246–270. Springer, 2001.

[16] Sven Koenig and Reid G. Simmons. Complexity analysis of real-time reinforcement learning. In *AAAI*, pages 99–107, 1993.

[17] Jaroslaw Koźlak, Jean-Charles Créput, Vincent Hilaire, and Abder Koukam. Multi-agent approach to dynamic pick-up and delivery problem with uncertain knowledge about future transport demands. *Fundam. Inf.*, 71(1):27–36, January 2006.

[18] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, July 2004.

[19] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:2005, 2005.

[20] Michele Piunti, Andrea Santi, and Alessandro Ricci. Programming soa/ws systems with bdi agents and artifact-based environments. In *Proceedings of Agents, Web Services and Ontologies, Integrated Methodologies Important Dates and Instructions (AWESOME-09)*, 2009.

[21] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, May 2009.

[22] S. Sen and G. Weiss. *Learning in multiagent systems*, pages 259–298. MIT Press, Cambridge, MA, USA, 1999.

[23] Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Stéphane Airiau. Learning context conditions for bdi plan selection. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '10, pages 325–332, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[24] Kornel Skakowski and Krzysztof Zieliski. Automatic adaptation of soa systems supported by machine learning. In LuisM. Camarinha-Matos, Slavisa Tomic, and Paula Graa, editors, *Technological Innovation for the Internet of Things*, volume 394 of *IFIP Advances in Information and Communication Technology*, pages 61–68. Springer Berlin Heidelberg, 2013.

[25] Bartomie Śnieżyński. Agent strategy generation by rule induction. *Computing and Informatics*, 32(5):1055–1078, 2013.

[26] Bartomiej Śnieżyński and Jacek Dajda. Comparison of strategy learning methods in farmerpest problem for various complexity environments without delays. *Journal of Computational Science*, 4(3):144 – 151, 2013.

[27] Bartomiej Śnieżyński, W. Wojcik, J. D. Gehrke, and J. Wojtusiak. Combining rule induction and reinforcement learning: An agent-based vehicle routing. In *Proc. of the ICMLA 2010*, pages 851–856, Washington D.C., 2010.

[28] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[29] Tomasz Szydlo and Krzysztof Zielinski. Adaptive enterprise service bus. *New Generation Comput.*, 30(2-3):189–214, 2012.

[30] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[31] Wojciech Turek, Andrzej Opalinski, and Marek Kisiel-Dorohinicki. Extensible web crawler - towards multimedia material analysis. In Andrzej Dziech and Andrzej Czyewski, editors, *Multimedia Communications, Services and Security*, volume 149 of *Communications in Computer and Information Science*, pages 183–190. Springer Berlin Heidelberg, 2011.

[32] Karl Tuyls and Gerhard Weiss. Multiagent learning: Basics, challenges, and prospects. *AI Magazine*, 33(3):41–52, 2012.

[33] M. Vuković. *Context Aware Service Composition*. PhD thesis, University of Cambridge, 2006.

[34] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[35] L.-J. Zhang, N. Zhou, Y.-M. Chee, A. Jalaldeen, K. Ponnalagu, R. R. Sindhgatta, A. Arsanjani, and F. Bernardini. Soma-me: a platform for the model-driven design of soa solutions. *IBM Syst. J.*, 47(3):397–413, July 2008.

[36] Krzysztof Zielinski, Tomasz Szydlo, Robert Szymacha, Jacek Kosinski, Joanna Kosinska, and Marcin Jarzab. Adaptive soa solution stack. *IEEE T. Services Computing*, 5(2):149–163, 2012.