An agent structure for evaluating micro-level MAS performance

Christos Dimou, Andreas L. Symeonidis and Pericles A. Mitkas
Aristotle University of Thessaloniki
Thessaloniki, Greece
cdimou@issel.ee.auth.gr, asymeon@issel.ee.auth.gr, mitkas@eng.auth.gr

Abstract — Although the need for well-established engineering approaches in Intelligent Systems (IS) performance evaluation is urging, currently no widely accepted methodology exists, mainly due to lack of consensus on relevant definitions and scope of applicability, multi-disciplinary issues and immaturity of the field of IS. Even existing well-tested evaluation methodologies applied in other domains, such as (traditional) software engineering, prove inadequate to address the unpredictable emerging factors of the behavior of intelligent components. In this paper, we present a generic methodology and associated tools for evaluating the performance of IS, by exploiting the software agent paradigm as a representative modeling concept for intelligent systems. Based on the assessment of observable behavior of agents or multi-agent systems, the proposed methodology provides a concise set of guidelines and representation tools for evaluators to use. The methodology comprises three main tasks, namely metrics selection, monitoring agent activities for appropriate measurements, and aggregation of the conducted measurements. Coupled to this methodology is the *Evaluator Agent Framework*, which aims at the automation of most of the provided steps of the methodology, by providing Graphical User Interfaces for metrics organization and results presentation, as well as a code generating module that produces a skeleton of a monitoring agent. Once this agent is completed with domain-specific code, it is appended to the runtime of a multi-agent system and collects information from observable events and messages. Both the evaluation methodology and the automation framework are tested and demonstrated in *Symbiosis*, a MAS simulation environment for competing groups of autonomous entities.

Keywords: *performance evaluation methodology, autonomous agents, multi-agent systems, automated evaluation*

I. Introduction

Evaluation is an integral part of any complete scientific or engineering methodology. Evaluation methodologies enable researchers to test the quality and applicability of their findings, as well as to set the limits and define the appropriate environmental or intrinsic parameters for optimal performance. The benefits of a well-defined evaluation methodology lead to detection of defects, safety and overall quality of a system. But, mainly, evaluation helps researchers to thoroughly comprehend the internal characteristics and impact of their newly proposed methods and ideas.

Although the need for generalized evaluation methodologies in the field on Intelligent Systems (IS) is indisputable, currently no such effort is known to the authors. This remarkable lack of means for evaluating the performance of intelligent systems may be attributed to a number of reasons. First, it is argued that IS technology has not yet reached a certain degree of maturity. Despite being in the center of attention for more than six decades, it is only recently that Artificial Intelligence (AI) and IS are applied to realistic problems. It is, thus, evident that more experience and time are needed in order to help this field reach the desired maturity level. Second, the research area of IS combines background theory and practices from a number of diverse scientific fields, including artificial intelligence, computational theory, distributed systems, even cognitive psychology and sociology. A coordinated course of action is therefore required, one that will integrate expertise gathered from all the above mentioned areas. Moreover, existing evaluation methodologies for conventional software do not suffice in the case of IS, due to the unpredictable performance properties that are not known at design time and may emerge at the execution of IS. Finally, there is a remarkable and possibly unresolvable lack of consensus on definition of relevant terms and scope of applicability of IS. It is, indeed, very difficult to define evaluation methods when there is no agreement on even the fundamental definitions, on what an intelligent system is, what constitutes an emergent behavior or what the scope of an IS should be.

Currently, researchers and developers that desire to evaluate their systems, often devise their own ad-hoc, domain-specific evaluation methods. It is often the case that these methods are biased (most of the times with no deliberation) so that they produce the optimal results for a very strict set of environmental parameters and assumptions. Moreover, their ad-hoc nature prevents third parties to repeat the experimental setting and verify the findings.

In this paper, we present a complete, generic, domain independent methodology for evaluating IS performance, as well as a supporting software tool that automates most of the evaluation process. The proposed methodology exploits the software agent paradigm as a representative modeling concept and implementation vehicle for intelligent systems. Indeed, agents may be regarded as entities that exhibit autonomous behavior in unknown and dynamic environments [11], being capable of encapsulating any existing intelligent technology, spanning from genetic algorithms [9], data mining [14] and machine learning [12], to reinforcement learning [6] and complex decision making techniques [5]. Agents rarely operate in isolation; they most often form groups or societies, either cooperating towards a common goal [13] or competing against each other on limited resources [17]. Thus, in multi-agent systems (MAS) [3] evaluators may focus on different levels of system granularity, ranging from single agent computational units and agent autonomy to multi-agent interaction, or even on complex global cooperation/competition aspects of MAS societies. The problem of performance evaluation is then reduced to three fundamental tasks, namely a) selection of appropriate metrics, b) monitoring agent activities for appropriate measurements, and c) aggregation of the conducted measurements. Within the context of this work, we address the above tasks, by providing a concise set of methodological steps and guidelines, as well as the corresponding agent structure that autonomously performs most of the monitoring workload.

The remainder of this paper is structured as follows: Section II reviews the current state-of-the-art in IS evaluation; Section III briefly presents the scope and basic concepts of our evaluation methodology; in Section IV, the automated evaluation tool, the *Evaluator Agent*, is presented; Section V applies the proposed methodology to *Symbiosis*, a MAS simulation environment for groups of autonomous entities and illustrates the results; Section VI concludes the paper and proposes some thoughts on future directions.

II. Related Work

Evaluation has been tightly coupled with artificial intelligence, since the early days of AI [15]. On their effort to define the capabilities and limits of machines, AI pioneers defined hypothetical evaluation benchmarks in order to compare potential computer behavior against the human intellect (e.g. [10]). The initial enthusiasm soon faded (during the infamous *AI winter*), giving room for traditional software evaluation, which focused on performance and quality assessment of conventional software products, as well as related productivity metrics. It is only recently, that IS have drawn once again the attention of computer scientists and engineers, this time with more realistic goals in specific engineering problems. However, as already mentioned, current evaluation efforts do not go beyond ad-hoc solutions.

There are two general approaches to the IS evaluation problem: the bottom-up and the top-down. The former, as elaborately represented by [19], advocates the definition of formal constructs and languages that will enable the definition of the appropriate terms and scope of IS. Evaluation will thereafter be gradually built upon these formal foundations. The latter approach observes that existing or newly implemented systems urge for evaluation methodologies and it is therefore preferable to instantly evaluate them at any cost. According to this approach, experiences from different ad-hoc evaluation attempts will be generalized into a concise domain-independent methodology, which in turn will be established at the time that IS reach a sufficient maturity level.

III. The Generalized Evaluation Methodology

Our generic evaluation methodology positions itself in compliance to the above-mentioned top-down approach. Motivated by the urging need to evaluate agent systems that are deployed with currently existing techniques, we provide a complete framework that will be readily available for developers. As a generic methodology, it could not take into account intrinsic implementation details, such as specific algorithms; instead it focuses on *observable* performance, which derives from system events and messages exchanged between the system modules and components. Moreover, this methodology addresses performance issues on different levels of granularity. It is the developer's choice to identify and focus on specific performance aspects at any level of detail, ranging from independent *computational units* of agents, to *autonomous agents*, *groups of agents*, or the entire *MAS*

With respect to each of the three basic tasks of the performance evaluation process (i.e. metrics, measurement and aggregation), our methodology provides either a theoretical representation tool with an accompanying set of guidelines or automated tools that assist evaluators throughout the process (see [2] for a detailed presentation of the methodology). More specifically:

1) *Selection of metrics*. Metrics are standards that define measurable attributes of entities, their units and their scope. Before any other decision, the evaluator must choose which attributes of the system he/she is interested in. For this purpose, we provide the Metrics Representation Graph (MRG), a hierarchical representation of metrics for a specific domain. Each node of MRG corresponds to a single metric. Leaf nodes represent directly measurable metrics (*simple metrics*), i.e. metrics that can be assigned with a specific measurement value. Measurable metrics, in turn, compose higher level metrics (*composite metrics*) that cannot be assigned with a specific value, but rather represent a higher level concept that is easier understood by the evaluator, using linguistic terms. For example, the simple metrics of *numberOfMessages* and *numberOfAgents* may be composed to produce the *scalability* composite metric. Traversing the hierarchy upward, one moves to higher level composite metrics, until one reaches the root, which is the total *systemEfficiency*. The evaluator is required to traverse the MRG and select only the metrics that are of interest to him/her. The general structure of MRG is provided in Figure 1.
2) *Measurement*. Measurement is the process of ascertaining the attributes, dimensions, extend, quantity, degree of capacity of some object of observation and representing these in the qualitative or quantitative terms of a data language [8]. Having selected the appropriate metrics, measurement is the next fundamental methodological step that systematically assigns specific values to these metrics. Typical measurement methods consists of experimental design and data collection. A measurement method is the answer to the question. Since measurement methods are difficult to summarize and categorize, we provide an automated tool for producing an Evaluator Agent that autonomously monitors the execution of a

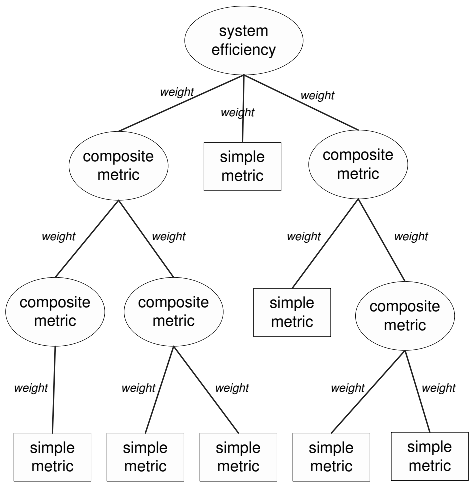

Fig. 1. General structure of the Metrics Representation Graph

system and records data related to the selected metrics. The Evaluator is presented in detail in Section IV.

3) *Fuzzy Aggregation*. Once the measurement procedure has been defined, the resulting metric-measurement pairs have to be aggregated to a single characterization for the investigated system. Aggregation, or composition, is the process of summarizing multiple measurements into a single measurement is such a manner that the output measurement will be characteristic of the system performance. Aggregation groups and combines the collected measurements, possibly by the use of weights of importance, in order to conclude to atomic characterization for the evaluated system. For example, an evaluated system may perform exceptionally well in terms of response time metrics (timeliness), but these responses may be far from correct (accuracy). Fuzzy sets [18] have been incorporated, since they provide efficient means of dealing with measurement of different scales and types, as well as of concluding to performance characterization, which is closer to the human language. Before aggregating the results, the evaluator needs to define appropriate fuzzy variables and membership function for the quantification of the measurement. He/she also has to define weights to each of the edges of the MRG, so that appropriate fuzzy aggregation operators may be later applied. These weights signify the importance of each sibling metric to the composition of the parent metric. At this moment, fuzzy quantification and weights assignment is done manually by the evaluator or a domain expert.

The complete set of steps that an evaluator is required to follow is summarized in Table I.

TABLE I

SUMMARIZATION OF METHODOLOGICAL STEPS

1.	Traverse MRG and select metrics
2.	Provide domain specific metrics (optionally)
3.	Determine metrics parameters
4.	Specify measurement method and parameters
5.	Execute experiments
6.	Define weights in the graph
7.	Define fuzzy variables and convert measurements accordingly
8.	Select and apply aggregation operators on the collected measurements

IV. EVALUATOR AGENT FRAMEWORK

In this section, we describe the general architecture and components of the *Evaluator Agent Framework*, a development framework for producing an agent structure that automates most parts of the proposed evaluation methodology. This framework is targeted to developers who need to evaluate the performance of their systems, either existing or under development. The basic requirements of this framework are the minimum modifications of the tested system at hand, as well as the minimum effort in writing evaluation specific code.

A. Purpose and Benefits

The purpose of the proposed framework is to guide the researcher/developer through the methodological steps of Table I by providing:

- visualization and manipulation of the MRG and the corresponding parameters of simple or composite metrics
- interactive specification of the fuzzy variables, membership functions and weights that correspond to the MRG
- automatic generation of the skeleton code of the *Evaluator Agent* that will undertake the task of monitoring the system for evaluation and collect all necessary information with respect to the selected metrics
- visualization of the performance evaluation results

At this point, a few of the abovementioned tasks still remain at the developers hands, as he/she is required to fill in the skeleton code of the Evaluator Agent with actual domain-specific parameters. Additionally, the developer is currently required to manually load and manipulate the MRG for the application domain at hand. At a latter phase, this will also be automated, as discussed in Section VI.

Applying the *Evaluator Agent Framework* to actual evaluation processes includes the realization of our primary motivation: the generalization of the evaluation process. By following a standardized step-wise approach for any given systems, two evaluators are expected to reach to the same conclusion. The framework will also reduce the time burden for evaluators, to tune parameters and manually monitor themselves the results. Additionally, readily available Application Programming Interfaces (APIs) for aggregating and visually presenting evaluation results may be used within any evaluation context. Fnally, this framework is envisioned to become a forum for collecting and utilizing knowledge for metrics from different domains. Developers within a domain-specific community will ideally be able to share MRGs and other experiences. Newcomers in any field will be provided with an MRG, accompanying weights and fuzzy sets from domain experts and will proceed to evaluation of their systems.

B. General Architecture

The general architecture of the proposed framework is depicted in Figure 2. The generic framework comprises five distinct components that are sequentially linked with user actions. The evaluation process initiates with the domain-specific MRG import action by the user. The MRG is presented through a interactive Graphical User Interface - GUI (Component A). After editing and manipulating the MRG, the user is provided with the skeleton of the *Evaluator Agent* (Component B). The user subsequently writes some domain specific code to produce the run-time Evaluator Agent and connects it to the system via the Evaluator Agent Middleware (Component C). The produced agent is then appended to the runtime of a MAS and starts monitoring and collecting measurement information (Component D). Upon user request or upon an end system event, the Evaluator Agent processes the logged information and produces graphs and other evaluation results (Component E), according to user preferences. Each of the components of the framework is further described in the next paragraph.

C. Components

In this subsection, the components of Figure 2 are further analyzed.

1) Component A - MRG GUI: This component is responsible for presenting the user with a visual representation of the MRG. The initiation of this process is done either by loading an existing MRG (which, ideally, is readily available from the domain communities) or by creating a new one. Node and edge manipulation tasks -such as edit and delete- are provided. For each simple or composite metric, the user may define a set of characteristic properties that are provided in drop-down menus and options. For example, for simple metrics, a user may define metric scales, units of measurement and metric types (e.g. range, boolean, nominal, ordinal etc). For composite metrics, the connection of a specific node to a set of other nodes (simple or composite metrics) with explicit edges declares that the parent node is composed of the children nodes. Through another MRG GUI option, users may also define weights for each edge, so that the participation importance of children metrics to the parent metric is determined. The set of weights will be later utilized in the fuzzy aggregation process.

After defining the structure and parameters of the MRG, the MRG GUI also provides a wizard for the determination of fuzzy variables that are necessary for the fuzzy quantification process. Appropriate fuzzy variables and corresponding membership functions are defined by the user and are correlated to each simple metric of the MRG. For example, a user may define the fuzzy variable *fastResponse* and correlate it to the simple metric *Response Time*. He/she must also provide a fuzzy membership function that maps actual measurements to the [0,1] range, as illustrated in Figure 3. It is evident that fuzzy variables and membership functions are heavily dependent on the application domain and are, currently, subjective specifications that are carried out manually by the user. The MRG GUI completes its execution by producing, upon user demand, the skeleton code of the Evaluator Agent.

The MRG GUI has been implemented as a plug-in for the Protégé Ontology Editor and Knowledge Acquisition System [4]. MRGs are represented in XML-RDF format and are loaded into the main Protégé platform. Readily available ontology visualization functions have been incorporated for the presentation of MRGs, and have been further enriched with metric-specific functionality for metric parameter, weight and fuzzy variable manipulation.

2) Component B - Skeleton Code Generator: Based on the fully defined MRG, the *Skeleton Code Generator* component is initiated in order to automatically produce the outline of an abstract, general Evaluator Agent, using the Java language. The resulting skeleton code consists of both complete and abstract classes. Complete classes implement the necessary infrasrtucture for processing, communication and logging tasks, whereas abstract classes are declared only to guide users through the domain specific addition to the Evaluator Agent.. Overall, the Skeleton Code Generator produces complete and abstract classes for:

- MRG specific manipulation functions
- metrics representation and processing functions
- tasks for collection of run-time data that correspond to the simple metrics of the selected MRG
- logging collected data into XML format
- processing XML log files
- aggregating measurement data
- communication primitives and message handling

The Skeleton Code Generator component essentially translates all the concepts that are represented in the MRG into specific or abstract code. If, for example, the user has selected metrics for the *agent level of granularity*, then the resulting code will be adjusted so that it efficiently addresses relevant single agent performance issues, such as accuracy, autonomy or timeliness. In a similar manner, if the user has selected the *MAS level of granularity*, the code will reflect performance issues, such as (possibly in addition to some of the above) scalability and modularity.

3) Component C - Evaluator Agent Middleware: The *Evaluator Agent Middleware* component serves as a connection API between the newly produced Evaluator Agent and the system under evaluation, as illustrated in Figure 4.

Since our evaluation methodology is based on the assumption that only observable behavior contains information on performance, the Evaluation Agent Middleware specifies the functions that manage observable events and observable messages, at different levels of granularity. The API provides functions for declaring, initiating, labeling and recording information on observable events and messages. Based on this API, the user is responsible to fill in the necessary code to the Skeleton Evaluator Agent, in order to produce a running Evaluator Agent. On the other hand, if it is not already implemented,

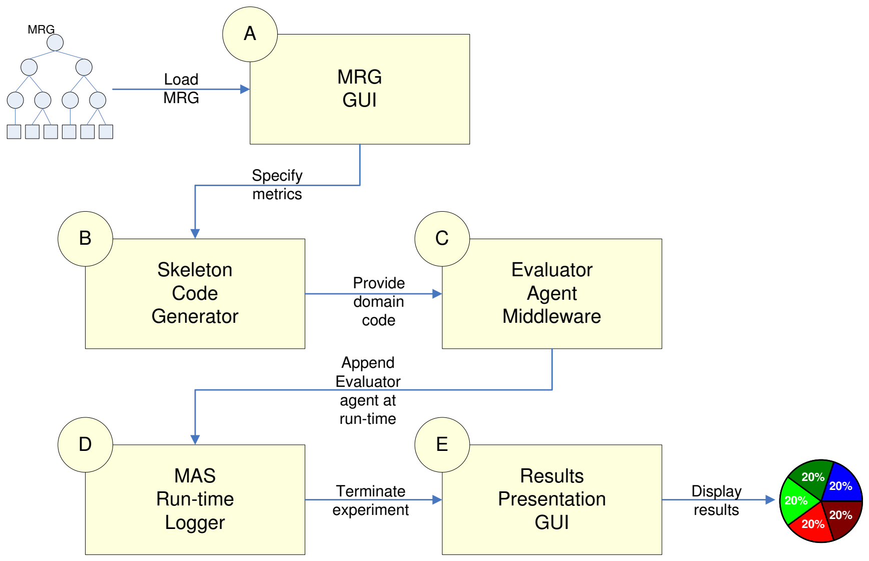

Fig. 2. Architecture of the Evaluator Agent Framework

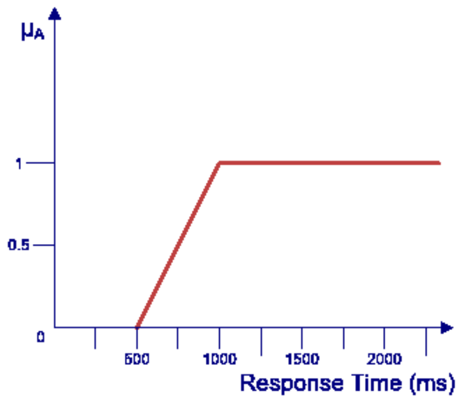

Fig. 3. Example fuzzy membership function for Response Time metric

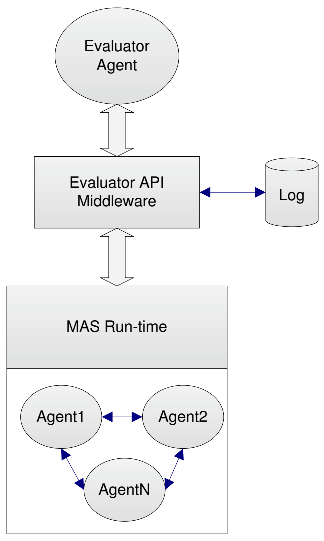

Fig. 4. Evaluator Agent Middleware

he/she may be required to provide some additional code to the original system in order to adhere to the Evaluator Agent Middleware. After the provision of the necessary code, the Evaluator Agent is ready to be appended to the run-time of the system.

The Evaluation Agent Middleware could be implemented in any existing programming language or platform. However, for testing purposes, we have implemented this component using the Java Agent Development Environment (JADE) [1]. Jade provides a comprehensive API for agent construction, behavior specification, communication management, as well as a few very useful tools, including the Sniffer API for tracking message content and events at runtime.

4) Component D - MAS runtime logger: This component undertakes the actual task of conducting the experimental measurement of the system's performance. The newly constructed Evaluator Agent participates as an observer agent in the MAS. On each declared event or sniffed message, the

Evaluator Agent calls the appropriate class or method in order to record performance-related information in XML format. This information may range from event or message timing to domain-specific parameter assessment, as for example the bid value in an electronic auction. The logging of this information is initiated at the designated *staring-event* and continues throughout execution until the designated *ending-event*. Both events, as well as iteration parameters, are defined by the user through the Evaluator Agent Middleware API.

5) Component E - Results Presentation GUI: For aggregation and presentation purposes, the *Results Presentation GUI* has been developed. This GUI loads one or more XML log files with all the performance information that has been recorded at runtime, as well as with the corresponding XML representation of the MRG. The user is then requested to select aggregation and presentation methods from a library of statistics, drawing, and fuzzy aggregation functions. Thus, for simple metrics a number of graphs and figures can be exported, while for composite metrics, fuzzy characterizations of parts or of the system as a whole are provided.

V. TEST CASE

In order to demonstrate the applicablity and validate the efficiency of the Evaluator Agent Framework, we have selected *Symbiosis* as a MAS for evaluation. In this section, a brief description is provided, and then we organize a selected set of metrics into a new instance of the MRG, apply the fuzzy quantification process and execute the experiments, using the generated Evaluator Agent.

A. Description of Symbiosis

Symbiosis [16] is mutli-agent simulation framework that follows the *animat* approach, as proposed by [7]. Animats represent autonomous, adaptive, learning entities that live and evolve in complex environments, in competition or collaboration with other animats. *Symbiosis* constitutes a virtual ecosystem, where two competing species of animats co-exist and share the environment's limited resources. Additionally, one of the two groups assumes the role of a group of *preys*, whereas the other is a group of *predators*. In addition to consumption of natural resources, predators may also consume preys. The goal of *Symbiosis* is to provide a simulation environment for testing and validating a number of emergent learning and adaptation techniques and the consequent effect of behavioral strategies.

The environment of *Symbiosis* is a $x \times y$ grid, where each cell can either be empty or occupied by:

- a natural resource, namely food, obstacle, trap
- a predator agent, or
- a prey agent

While natural resources are static, preys and predators are free to move in any neighbouring cell, aiming to maximizing their energy, either by visiting energy enhancing cells (food cells for preys and prey cells for predators) or by avoiding energy reducing cells (predator cells for preys and obstacles and traps for both species). Each agent is born with an initial amount of energy, certain vision and communication capabilities, a decision-making mechanism and reproduction abilities. The decision-making mechanisms employs genetic algorithms for the classification and evaluation of a set of action rules, based either on previous experiences or communicated from a neighbouring entity of the same species. Finally, in order to reproduce conditions that occur in real-world environments, uncertainty hs been introduced in *Symbiosis*, in the form of a parameterised vision error probability.

B. MRG Instance

The experimental measurement of the *Symbiosis* performance is based on a set of simple, measurable metrics that have been proposed and analyzed in [16]. These metrics are:

- *energy (en)*: the energy balance of an agent
- *age (ag)*: the number of *epochs* an agent lives
- *resource consumption rate (rcr)*: the ratio of energy enhancing cells that an agent has visited, to the number of total moves
- *trap collision rate (tcr)*: the ratio of trap cells that an agent has collided upon, to the number of total moves
- *unknown situation rate (usr)*: the ratio of the total unknown situation (no suitable rule applied), to the number of total moves
- *reproduction rate (rr)*: the ratio of the total offspring of an agent, to the number of total moves
- *effectiveness (e)*: the energy uptake rate minus the energy loss rate, to the energy availability rate
- *net effectiveness (e_{NET})*: the effectiveness of preys, without taking into account losses caused by their interaction with predators

After specifying the simple metrics, we need to aggregate them into composite metrics that are more comprehensible by the human evaluator. The resulting MRG is depicted in Figure 5. At depth *2*, we have concluded on the following composite metrics: *adaptability*, *scalability*, *durability* and *robustness*. All these metrics are further composed to produce both *preyEfficiency* and *predatorEfficiency* at depth *1*. Finally, the root composite metric, *MASEfficiency* is naturally composed by the two aforementioned metrics, with the addition of the *stability* simple metric, which corresponds to the deviation of total energy imbalance between the two species.

It must be noted that the weights presented in Figure 5 have been determined by a domain expert and are therefore subjective.

C. Fuzzy quantification

The next step is to determine a fuzzy variable and the corresponding membership value for each of the simple metrics defined in the previous paragraph. For simplicity and conciseness, we assign a **_high* fuzzy variable for each simple metric, where * is the name of the metric. For example, the fuzzy variable for *rcr* is named *rcr_high* and implies a high degree of resource consumption rate. Based on the knowledge provided by the domain expert, we have defined the membership function for *rcr_high*, which is depicted in

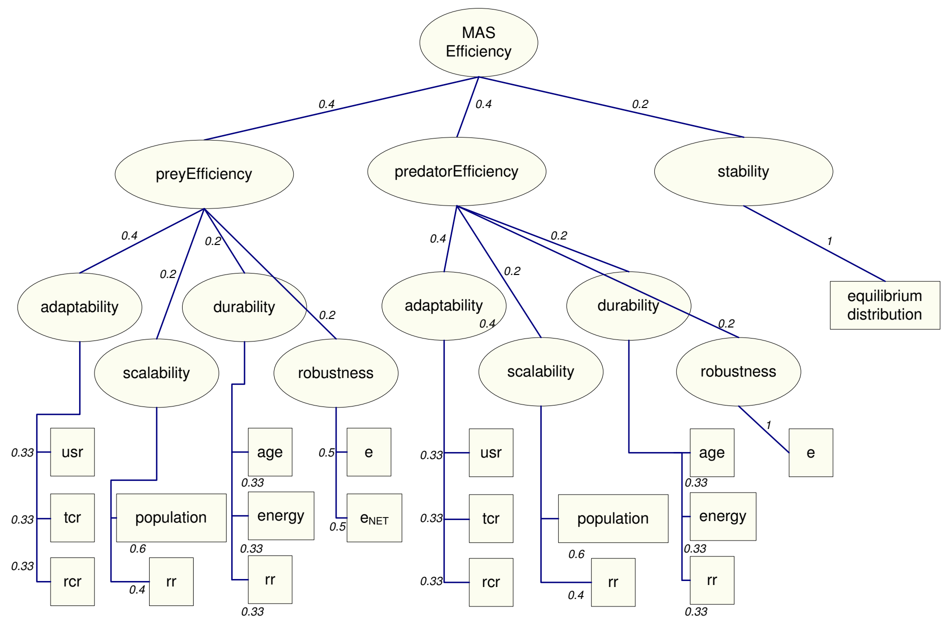

Fig. 5. An instance of the MRG for Symbiosis

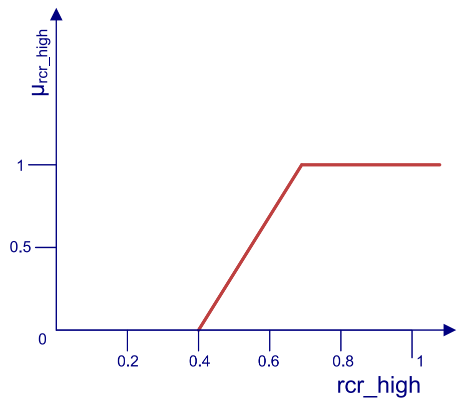

Fig. 6. Fuzzy membership function for rcr_high

Figure 6. We follow a similar approach for the rest of the simple metrics.

D. Experiments

Having completed the above steps, we continue with the automatic generation of the Skeleton Code and the actual implementation of the Evaluator Agent. Since the content of the exchanged messages between agents is not of importance to performance, the resulting agent is restricted to observe and record designated events, including food (or prey) consumption, trap collision, reproduction and unknown situations. The only burden assigned to the developer was to modify the original code in order to trigger new events in the above situations.

Two series of experiments were conducted in order to asses performance issues related to the behavior of the animats with respect to the classification mechanism and the environmental variety, respectively. In the first series of experiments, for certain environmental parameters, the impact of the classification mechanism to the system efficiency was examined. For varying values of the employed genetic algorithm invocation step (in the range of [50,500]), each of the selected simple metrics was measured. After the fuzzification and aggregation, it was determined that the highest system efficiency is a result of the genetic alogirhm step that equals to 100.

The above optimal value was then provided to the second series of experiments, which focus on prey's efficiency. A taxonomy of environments was created, as described in more detail in [16]. For each of these environments, the preys used the classification mechanism with the optimal value for the genetic algorithm step, whereas the predetators either used the same mechanism or did not use any learning mechanism at all. It was easily confirmed that Experiment B7 of the original paper provided the best results for *preyEfficiency* in total, as well as for *adaptability*, *durability* and *robustness* metrics.

Overall, the testing of the proposed evaluation methodology and the Evaluator Agent proved to be useful for carrying out preformance evaluation tasks for an already developed system. As expected, the results of this evaluation process adhere to the experimental findings of the original paper, a fact that was a principal goal of our system. Moreover, the performance of the system was analyzed in many more composite metrics, that were examined and compared in isolation of the rest of the system. This way, the evaluator may easily identify defective parts or modules of his/her system that affect the performance of the entire system. The only shortcoming of the entire experimentation process was the fact that the presence of a domain expert (in our case, the developer of the system) was necessary, both for the definition the MRG, as well as the provision of domain specific code for the Evaluator Agent.

VI. Conclusions

Driven by the urging need to provide general methodologies and tools for IS performance evaluation, we presented a novel methodology and accompanying tools for evaluating the unpredictable, emerging behavior of agents and MAS in dynamic environments. The proposed methodology provides concise methodological steps, which the evaluators may follow to guarantee a standardized and repeatable evaluation procedure. Focused on the observalble aspects of agent behavior, such as messages and events, the methodology provides representation tools for organizing, categorizing and aggregating performance metrics. Fuzzy sets have been incorporated to represent higher-level composite metrics that are more meaningful to the human evaluator.

The *Evaluator Agent Framework* was also described and demonstrated. The goal of this framework is to automate most of the steps of the above methodology, by providing GUIs for metrics and results manipulation, as well as a code generating component for automatic monitoring of observable behaviors at runtime. The produced *Evaluator Agent* monitors messages and events, while recording all performance related information for posterior processing. The Evaluator Agent Framework was successfully tested on *Symbiosis*, a MAS simulation framework for adaptive autonomous agents.

The most important future direction that emerges from this work is the automation of the MRG definition process. Currently, domain experts are being mobilized to specify the simple and composite metrics, the corresponding fuzzy variables, the fuzzy membership values and finally the weights in the MRG. Some of these tasks may be automated by exploiting information on previous evaluation efforts and historical data. It is also feasible to use this information as training datasets in order to train predefined, domain-specific MRGs. It is our vision that this effort will initiate the sharing of domain knowledge, metrics and practices towards more standardized evaluation procedures.

Acknowledgment

This paper is part of the 03ED735 research project, implemented within the framework of the “Reinforcement Programme of Human Research Manpower” (PENED) and co-financed by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Funding).

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with jade. In *Eleventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, Boston, MA, USA, 2000.

[2] C. Dimou, A. Symeonidis, and P. Mitkas. Evaluating knowledge intensive multi-agent systems. In *Proceedings of the Autonomous Information Systems - Agents and Data Mining Conference*, St. Petersburg, Russia, 2007.

[3] W. G. *Multiagent systems. A modern approach to distributed artificial intelligence*. The MIT Press, 1999.

[4] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubzy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of protg: An environment for knowledge-based systems development.

[5] N. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

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

[7] F. Krebs and H. Bossel. Emergent value orientation in self-organization of an animat. *Ecological Modelling*, 96(1):143–164, 1997.

[8] K. Krippendorff. *A Dictionary of Cybernetics*. The American Society of Cybernetics, Norfolk, VA, USA, 1986.

[9] F. Menczer, W. Street, and M. Degeratu. Evolving heterogeneous neural agents by local selection. In V. Honavar, M. Patel, and K. Balakrishnan, editors, *Advances in the Evolutionary Synthesis of Neural Systems*. MIT Press, Cambridge, MA, 2000.

[10] A. Newell and B. Buchanan. Artificial intelligence. *Encyclopedia of Science and Technology*, 2(1):146–150, 1997.

[11] H. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11:1–40, 1996.

[12] T. R. Payne and P. Edwards. Learning mechanisms for information filtering agents. In J. L. Nealon and N. S. Taylor, editors, *Proceedings of the UK Intelligent Agents Workshop*, pages 163–183, Oxford, 1997. SGES Publications.

[13] V. Renganarayanan, A. Nalla, and A. Helal. Internet agents for effective collaboration, 2001.

[14] A. Symeonidis. *Agent Intelligence through Data Mining*. Springer Science and Business Media, 2005.

[15] A. Turing. Computing machinery and intelligence. *Mind*, 59(1):443–460, 1950.

[16] F. Tzima, A. Symeonidis, and P. Mitkas. Symbiosis: Using predator-prey games as a test bed for studying competitive co-evolution. In *Proceedings of the Knowledge Intensive Multi-Agent Systems (KIMAS-07)*, Boston, MA, USA, 2007.

[17] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.

[18] L. Zadeh. Fuzzy sets. *Information and Control*, 8(1):338–353, 1965.

[19] L. A. Zadeh. In quest of performance metrics for intelligent systemsa challenge that cannot be met with existing methods. In *Proc. of the Third International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 13-15 August 2002.