Data Mining and Agent Technology: a fruitful symbiosis

CHAPTER · DECEMBER 2007

DOI: 10.1007/978-0-387-69935-6_14

3 AUTHORS, INCLUDING:

Andreas Symeonidis
Aristotle University of Thessaloniki
79 PUBLICATIONS **336** CITATIONS
SEE PROFILE

Pericles A. Mitkas
Aristotle University of Thessaloniki
250 PUBLICATIONS **1,372** CITATIONS
SEE PROFILE

Data Mining and Agent Technology: a fruitful symbiosis

Christos Dimou[1], Andreas L. Symeonidis[1,2], and Pericles A. Mitkas[1,2]

[1] Electrical and Computer Engineering Dept.
Aristotle University of Thessaloniki, 54 124, Thessaloniki, Greece
[2] Intelligent Systems and Software Engineering Laboratory,
Informatics and Telematics Institute/CERTH, 57 001, Thessaloniki, Greece
cdimou@issel.ee.auth.gr, asymeon@iti.gr, mitkas@eng.auth.gr

1 Introduction

Large amounts of data are being produced and made available online every day, pushing user needs towards a more knowledge-demanding direction. Today's applications are therefore required to extract knowledge from large, often distributed, repositories of text, multimedia or hybrid content. The nature of this quest makes it impossible to use traditional deterministic computing techniques. Instead, various soft computing techniques are employed to meet the challenge for more sophisticated solutions in knowledge discovery. Most notably, Data Mining (DM) is thought of as one of the state-of-the-art paradigms. DM produces useful patterns and associations from large data repositories that can later be used as *knowledge nuggets*, within the context of any application.

Individual facets of knowledge discovery, introduced by DM techniques, often need to be orchestrated, integrated and presented to end users in a unified way. Moreover, knowledge has to be exploited and embodied in autonomous software for learning purposes and, hence, a more increased performance (Figure 1). Agent Technology (AT) proves to be a promising paradigm that is suitable for modelling and implementing the unification of DM tasks, as well as for providing autonomous entity models that dynamically incorporate and use existing knowledge. Indeed, a plethora of multi-agent systems (MAS) and other agent-related solutions for knowledge-based systems can be found in the literature, and more specifically in the area of agent-based DM, as it is explained in detail in Section 3 of this chapter.

A numerous related agent development methodologies deal with most of the steps of the development lifecycle. However, there is a remarkable lack of generalized evaluation methodologies for the systems in question. The evaluation of performance is a fundamental step of any development methodology, which provides developers with countable, qualitative and verifiable attributes

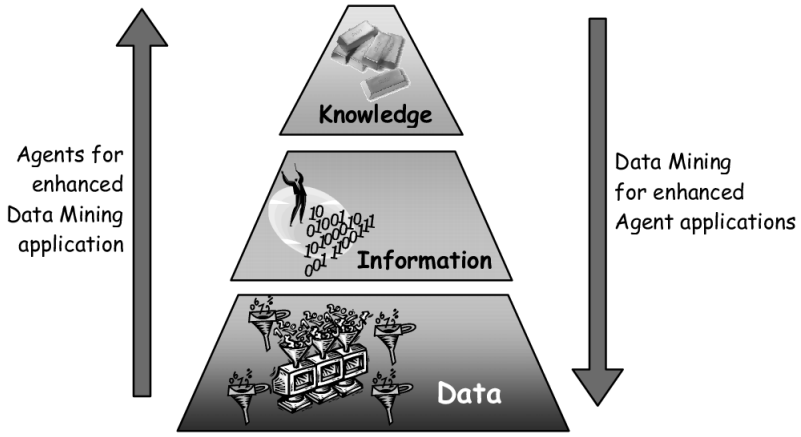

Fig. 1. Mining for intelligence

in an effort to better comprehend the nature of the system at hand. Additionally, generalized and standardized evaluation procedures allow third parties to safely verify the acclaimed properties of deployed systems or newly discovered scientific results.

Existing evaluation approaches address either the DM algorithmic issues or the overall system performance. Both approaches come short in the case of AT and DM integration, due to the complex and dynamic nature of the produced systems. In the case of DM evaluation, focus is given on the statistical performance of individual techniques, in terms of precision and recall, ignoring the actual impact of the extracted knowledge to the application level. In the case of overall system evaluation, existing methods fail to deal satisfactorily with emergent agent behaviors that may not be known at design time.

In this chapter, we present an integrated software engineering approach for developing DM-enriched MAS. Having *Agent Academy II* as the basic designing, development and agent training framework for MAS that employ DM, we provide a generalized methodology for evaluating the performance of a developed system. A set of consice methodological steps is presented, focusing on three fundamental evaluation aspects, namely the selection of a) metrics, b) measurement method, and c) aggregation methods. The proposed methodology is designed to assist developers as an off-the-shelf tool that can be integrated in the overall system development methodology.

The remainder of this chapter is organized as follows: in Section 2 an overview of the basic primitives of AT and DM is provided; Section 3 reviews the related literature in DM and MAS integration and evaluation; Section 4 presents the core development and evaluation methodology, by outlining the appropriate theoretical and software tools; in Section 5, Agent Academy II, a

development framework for DM-enriched MAS is presented; finally, Section 6 summarizes and discusses related concluding remarks.

2 Agent Technology and Data Mining

AT and DM have been incorporated and integrated in numerous research efforts. However, the disparity of applications and the notable diversity of the nature of these technologies motivate us to provide thorough definitions of relevant terms and present our point of view over their symbiosis.

2.1 Agents

The term "software agent" has been coined since the early years of Artificial Intelligence, in order to denote any software module that exhibits intelligent behavior. This vague definition, in combination with the unfeasible visions of early AI, has resulted into unsatisfactory applications and has decreased agent computing popularity for many years. It is only until recently that interest in software agents has revived within the context of complex systems' engineering. Agents appear as a handy modelling concept for autonomous, decentralized entities, cooperating towards a common goal or competing on limited resources. Moreover, the advent of many popular, highly distributed internet applications has reinforced agents' position as a promising paradigm for addressing the emerging engineering problems [45][25][15][20].

In practice, however, no single universally agreed-upon definition of a software agent exists. This problem occurs due to the horizontal nature of agents. Agents can be either abstract tools for modelling complex systems or actual implemented software modules that may perform any task. The vast disparity of application domains on which agents have been applied reinforces the definition difficulties. In some sets of applications agents may need to exhibit decision making behaviors, whereas in other cases agents may be assigned routine, predefined tasks. Various abstract definitions for agents have been proposed, focusing on one or more agent characteristics with respect to one or more application domains. Woolridge and Jennings, for example, define a software agent with respect to situatedness, autonomy and goal-orientation.

In general, an agent is a software entity that exhibits some or all of the following characteristics:

1. *Autonomy*: Considered as one of the most fundamental features of agency, autonomy implies the degree of control that an agents poses on its own execution thread. Autonomy is usually a strong requirement in many application domains and therefore agents often employ relevant techniques for task-wise decision making in their effort to accomplish their goals.
2. *Interactivity*: Agents are seldom stand alone. They most often rely in information rich and eventful environments with other agents, services or

human users. It is therefore required for agents to possess corresponding sensors for perceiving information and actuators for changing the environment. Moreover, interactivity emerges as a result of a more complex intrinsic behavioral processing of the environmental input. Therefore, agents may either respond to occurring events (*reactiveness*) or take initiative and act in order to accomplish predefined goals (*proactiveness*).

3. *Adaptability*: In dynamically changing environments, agents need to be able to change their internal states and consequent actions, to better match the ever-changing conditions.
4. *Sociability*: A product of interactivity, socialability is based on relative human social skills, such as the ability to determine trusted parties or form coalitions in unknown environments.
5. *Cooperativity*: In applications such as distributed problem solving, agents are often required to collaborate with each other in order to reach a common goal that otherwise would be impossible or impractical to reach. Cooperation may be coordinated by dedicated agents or, in more open environments, emerge via agent communication.
6. *Competitiveness*: Agents are often programmed to allocate certain resources in well-defined environments. Like in similar real-world scenaria, the resources are limited and therefore agents need to compete against each other in order to allocate such resources. Agents employ strategies and action plans for prevailing in such competitive environments.
7. *Mobility*: Transition between different environments is a desired property of agents in applications such as data gathering, web crawling etc.
8. *Character*: Human-centered characteristics can be summarized as a character that is embodied in agents, most often in interface, personal, or assisting agents.
9. *Learning*: Learning is an all-encompassing term that utilizes some or all of the above properties, so that agents observe the impact of theirs or other agents' actions on the environment and predict the optimal behavior, that is activated in similar situations in the future.

Alternative definition approaches attempt to classify agents with respect to their application domain. Such domains may include *inter alia* searching and filtering of information, monitoring conditions, alerting, proxing, coordinating network tasks and managing resources.

A robust classification of agents is provided by [36]. According to this approach, agents can be classified with respect to the following dimensions:

1. *Mobility*, that differentiates agents into *static* and *mobile*
2. The *logic paradigm* they employ, which classifies them as either *deliberative* or *reactive*
3. The *fundamental characteristic* that describe the agent (autonomy, cooperativity, learning). Based on these axes, agents can be classified as (Figure 2):
   - collaborative agents

- collaborative learning agents
- interface agents
- smart agents

In this work, we have adopted Nwana's classification scheme, since it covers successfully a wide area of agent-related applications, as well as it proves to be robust enough to meet the needs of a large number of researchers in the AT field.

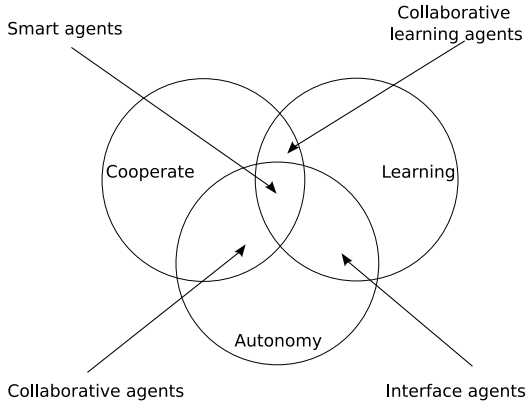

Fig. 2. Nwana's classification of agents

2.2 Multi-Agent Systems (MAS)

The promising properties of agents discussed above can only be fully exploited in complex architectures that are deployed in a systematic way. *Agent Oriented Software Engineering* (AOSE)[26][37] tackles this challenge by providing high level abstraction, modelling and development tools, grouped under the umbrella term of Multi-Agent Systems (MAS). A MAS is as a systematic solution to inherently distributed problems by utilizing autonomous interacting agents and appropriate communication protocols. MAS integrate the above mentioned agent properties and provide new system features that would otherwise be impossible to achieve using monolithic systems. Such characteristics include:

- Increased *performance*, *reliability* and *maintenability*.
- *Modularity*, *flexibility* and *extensibility*.
- Overcome of *limited scope* and *capacity* of single agents in terms of knowledge and tasks.
- Better support for *dynamic*, *unpredictable environments*.

- Introduction of *collective intelligence*, that is the augmentation of intelligence of many simple, not-so-smart interacting entities in contradiction to single, centralized sophisticated modules.

Distributed systems, in the above sense, are modelled as a network of autonomous entities that regulate, control and organize all activities within the distributed environment. In the recent years, a plethora of such distributed systems has emerged. Most notably, the Grid paradigm [14] that envisions a transparent infrastructure of high performance computer resources or knowledge resources that is scattered throughout the globe. Since the conceptualization of the Grid, it has become apparent that MAS may play a pivotal role in the modelling and implementation of such systems[15][44]. Other agent-based popular applications include Web Services[17][33][34], Web Crawling[24][9], environmental monitoring[4][38] and virtual market places[50][46].

MAS Characteristics

In general, MAS adhere to the following three primitives. First, MAS must specify appropriate communication and interaction protocols. Despite agents being the building blocks of a problem solving architecture, no individual problem could be effectively solved if no common communication ground, and no action protocol exists.

Secondly, MAS must be open and decentralized. No prior knowledge of, for example, number of participants or static behaviors are always known to the system developer. In a running MAS, new agents may join at any time having only to comform to the communication protocol, being able to act on the way they choose, often in unpredictable manner.

Finally, MAS must consist of possibly heterogeneous agents that are scattered around the environment and act autonomously or in collaboration.

2.3 Data Mining

The need for methods for discovering useful information in large data volumes has been a vivid research topic for many years. Especially nowdays, this need is imperative due to the increasing rate of data production, intensified by the ever-increasing demand for information. DM is a relatively new approach to this problem, often denoted as Knowledge Discovery in Databases (KDD), deals with this exact problem: "the extraction of interesting, non-trivial, implicit, previously unknown and potentially useful information or patterns from data in large databases"[10]. Though a large number of patterns may arise from the application of DM on datasets, only interesting ones are selected, that is patterns that can be easily understood by humans and suitable for validating a user hypothesis.

Other researchers argue that DM is only "the most important step in the KDD process and involves the application of data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce

a particular enumeration of patterns of data"[2]. Either approach adopted, in essence DM and KDD address the same problem of extracting useful knowledge and, within the context of this chapter, integrate this knowledge into MAS.

KDD process

KDD is the iterative traversal of the list of steps presented in Table 1

Table 1. Steps of the KDD process

1. Identify the goal of the KDD process
2. Create a target dataset
3. Clean and preprocess data
4. Reduce and project data
5. Identify the appropriate DM method
6. Select a DM algorithm
7. Apply DM
8. Evaluate DM results
9. Consolidate discovered knowledge

One or more steps of the KDD process may be repeated as many times deemed necessary, in order to come up with desirable outcome.

Data Mining Techniques

DM techniques may be applied on large data sets either for validation of a hypothesis or for discovering new patterns. The latter case is further divided into prediction of future trends of the data fluctuation and to a more detailed description and understanding of already extracted patterns. Within the scope of this chapter, focus is given on discovering new patterns and the associated DM techniques include[10]:

1. *Classification*: the discovery of knowledge model that classifies new data into one of the existing pre-specified classes.
2. *Characterization and Discrimination*: the discovery of a valid description for a part of the dataset.
3. *Clustering*: the identification of finite number of clusters that group data based on their similarities and differences.
4. *Association-Correlation*: the extraction of association rules, which indicate cause-effect relations between the attributes of a dataset.
5. *Outlier analysis*: the identification and exclusion of data that do not abide by the behavior of the rest of the data records.
6. *Trend and evolution analysis*: the discovery of trends and diversions and the study of the evolution of an initial state/hypothesis throughout the course of time.

2.4 Integrating Agent Technology and Data Mining

AT and DM are two separate vessels that each, as derived from the above, has its own scope and applicability. The idea of combining these diverse technologies is emerged by the need to either a) enrich autonomous agents by employing knowledge derived from DM or b) utilize software agents to assist the data extraction process. Both aspects are intriguing and challenging, mainly because of the disparity of AT and DM, since:

1. Despite logic being their common denominator, AT and DM employ two complementary logic paradigms. Agent reasoning is usually based on deductive logic, whereas DM embraces inductive logic.
2. Categorization of MAS with respect to DM-extracted knowledge is complicated due to the wide application range of both technologies.

Logic paradigms

In deductive inference, conclusions are drawn by the combination of a number of premises. Thus, knowledge models are applied to data producing knowledge information[13]. Under the assumption that these premises are true, deductive logic is truth preserving. In MAS applications, deduction is used by agents in a form of predefined rules and procedures usually defined by domain experts. These rules specify agent actions with respect to the input sensed. Nevertheless, deduction proves inefficient in complex and versatile environments[3][48].

Inductive inference, on the other hand, attempts to transform specific data and information into concrete, generalized knowledge models. During the induction process, new rules and correlations are being produced aiming at validating each hypotheses. In contradiction to deduction, induction may lead to invalid conclusions, as it only uses progressive generalizations of specific examples[29].

It becomes evident that the coupling of the above two approaches under the combination of the carrying technologies leads to enhanced and more efficient reasoning systems as proved by [42]. Indeed, this combination overcomes the limitations of both paradigms by using deduction for well-known procedures and induction of discovering previously unknown knowledge. The processes of agent training and knowledge diffusion are further explained in the remainder of this section.

Agent modelling

Knowledge extraction capabilities must be present in agent design, as early as in the agent modelling phase. During this process, the intended DM techniques employed shape the nature of the reference engine and provide the knowledge model of the agent with required useful patterns. Every agent exhibiting such reasoning capabilities is required to have the internal structure outlined below.

- Application domain
- Domain ontology
- Agent shell
- Agent type
- Behavior type
- Knowledge Model
- Reasoning engine

When the DM-generated knowledge model is incorporated to the otherwise dummy agent, the outer layers, namely domain ontology, agent shell, agent type and behavior type consist the functional parts of the agent that are influenced in corresponding manners.

The process of dynamically incorporating DM-extracted knowledge models (KM) into agents and MAS is defined as "agent training" while the process of revising in order to improve the knowledge model of agents by reapplying DM techniques is defined as retraining. Finally, "knowledge diffusion" is defined as the outcome of the incorporation of DM extracted knowledge to agents.

Three levels of diffusion

Knowledge diffusion is instantiated in three different ways, with respect to the alternative targets of DM:

1. *DM on the application level of MAS.* DM is applied in order to find useful rules and associations in application data, in order to provide knowledge nuggets to the end user, independently of the internal architecture.
2. *DM on the behavioral level of MAS.* DM is applied on behavioral data, usually log files with past agent actions, in order to predict future agent behaviors.
3. *DM on the evolutionary agent communities.* Evolutionary DM is performed on agent communities in order to study agent societal issues. According to this societal point of view, the goals that need to be satisfied are not atomic but collective and, therefore, knowledge is fused into agents that share common goals.

3 Related work

3.1 MAS and DM

The integration of AT and DM is a subject of a number of research efforts that can be found in the literature. In [16], a combination of inductive and deductive logic for reasoning purposes is proposed for improved customer relationship management. In this work, deduction is used when complete information is available, whereas induction is employed to forecast behaviors of

customers when the available information is incomplete. [13] provides an implementation of a single inference engine for agents that uses both inductive and deductive reasoning. In this work, logic terms of model data, information and knowledge are incorporated and processed by deductive agents. Finally, an integration of deductive database queries and inductive analysis on these queries and their produced knowledge is presented in [27].

It is a frequent observation in MAS applications that a tradeoff between inference (either inductive or deductive), complexity and development cost arises. However, in more dynamic environments, where both requirements and agent behaviors need constant modification, a systematic approach is compulsory. Symeonidis and Mitkas [42] present a unified methodology for transferring DM extracted knowledge into newly created agents. Knowledge models are generated through DM on the various levels of knowledge diffusion and are dynamically incorporated in agents. The iterative process of retraining through DM on newly acquired data is employed, in order to enhance the efficiency of intelligent agent behavior. The suggested methodology is thoroughly tested on three diverse case studies.

3.2 Evaluation

Although promising, the integration of DM results into MAS functionality arises interesting and some times crucial issues, as far as safety and soundness is concerned. Seeking to extend the work of [42] and provide an evaluation framework for agent efficiency, we present a literature review on intelligent agent evaluation.

Evaluation is a vital step in any complete scientific and engineering methodology. It is defined as "the process of examining a system or a system component to determine the extend to which specified properties are present"[1]. It is the most powerful and sound tool for researchers to assess the quality and applicability of their findings, as well as to set the limits and the optimal environmental or intrinsic system parameters for optimal performance.

Within the Soft Computing paradigm, evaluation is an all encompassing term that may address any component of the system at hand, heavily depending on the developer's aims. It is therefore the researcher's choice to focus on: a) algorithmic issues, b) system performance evaluation, or c) observable intelligence of the implemented system. By addressing one or more of the above, a researcher is able to isolate theoretical shortcoming or implementation malpractices, comprehend the intrinsic characteristics of the system and improve it in the most beneficial manner.

Algorithmic performance and quality evaluation in the context of Soft Computing has been an issue covered by a large corpus of work in the literature, that especially draws from the information retrieval theory primitives.

[1] The Free Online Dictionary of Computing, September 2003 (http://www.foldoc.com)

In such cases, the algorithm employed is assessed against its ability to extract the largest percent possible of useful information. Common metrics in this direction include precision, recall, fallout, F-measure and mean average precision. Other Soft Computing approaches use other appropriate metrics or aggregation techniques, depending on the case, including ROC curves, fitness functions and composite figure of merits. Algorithmic evaluation is an important tool that gives a summary of the black box implementation of any algorithm.

In the case of intelligent systems, such as MAS, instead of evaluating individual algorithms, we need to assess the actual impact of the employed techniques with respect to other engineering aspects, such as integration issues, performance issues and impact of the selected techniques to the overall quality of the outcome. Moreover, emergent, unpredictable and intelligent behavior that often is exhibited by such systems complicates the process of defining and realizing evaluation procedures. For instance, a bidding agent may use a DM knowledge extraction algorithm from historical data. Despite the satisfaction of high algorithmic precision, the agent may end up losing all auctions because of inefficiency in timing and/or overall strategy. We, therefore, need to regard the system at hand as an integrated intelligent system that consists of modules, exhibits certain behaviors and aims at the accomplishment of specific goals.

In the literature, two general research approaches towards the direction of engineering aspects evaluation exist: a) bottom-up and b) top-down. The first approach represents the strong AI perspective on the problem, indicating that intelligent systems may exhibit any level of intelligence comparable to human abilities. Zadeh [51] argues that evaluating such systems is infeasible today, due to the lack of powerful formal languages for defining intelligence and appropriate intelligent metrics. The second approach represents the weak AI or engineering perspective, according to which intelligent systems are systems of increased complexity that are nevertheless well-defined in specific application domains, designed for solving specific problems. Albus [1] suggests that intelligent performance can be effectively evaluated after a concise decomposition of the problem scope and definitions of relative metrics and measurement procedures. Driven by the urging need to evaluate and compare existing or emergent applications, we adopt the top-down approach.

It should be denoted at this point that no general, complete methodology for evaluating engineering applications exists. Instead, researchers often have to devise their own ad-hoc metrics and experimental procedures. In fact, in some cases, the chosen parameters or input data are chosen so as to produce the best results for the -each time presented- method. Moreover, the findings are often supported by qualitatively arguments only, in favor of the proposed system and no debate with respect to its drawbacks is provided. Consequently, it is impossible for a third party to repeat the evaluation procedure and validate the quality of the proposed solution by concluding to similar results. The need for a generalized evaluation framework is, thus, evident.

The requirements of such a methodology is to address both system performance issues as well as emergent, intelligent atomic and social behavior. To answer this challenge, we must devise a methodology that focuses on the following:

- *Re-usability*: The provided methodology must be domain independent and must be available for reuse under different application scenaria, always taking into account possible inherent or emergent heterogeneity in different implementations.
- *Qualitative comparability*: Different implementations in a specific application domain must be liable to comparison with respect to a set of selected qualitative features.
- *Quantitative assessment*: Different implementations in a specific application domain must also be liable to comparison with respect to a set of selected quantitative criteria. Additionally, there must be the opportunity of defining optimal or desired performance, against which one or more implementations may be compared.

Software engineering evaluation, as a well established field, consists a major source of background theory and tools towards this direction. Complete methodologies with quantitative and qualitative metrics have been developed and used in actual software projects. Although subsets of metrics and methods may be adopted, these approaches do not suffice for evaluating intelligent systems, since standard software evaluation processes focus in product features and do not always take into account emergent and unpredictable system behavior.

Ongoing efforts for generalized metrics and evaluation methodologies exist in application fields, such as robotics and autonomic computing. In robotics, evaluation efforts span from autonomous vehicle navigation [35][22][52] to hybrid human-robot control systems[39][6][18]. In autonomic computing, emphasis is given to the quality assessment of the selected self-managing techniques [23]. Both fields provide us with usefull metrics and thorough methodological steps. However, neither of the above approaches are complete and mature nor do they provide us with relevant tools for the case of knowledge infusion in autonomous entities.

4 A methodology for designing and evaluating DM-enriched applications

In this section we present a generalized methodology for comprehensive development of DM-enriched Multi-Agent Systems. While a typical designing methodology comprises many parts, we focus mainly on the evaluation part of our methodology, for two reasons. First, a multiplicity of designing methodologies exists in the area of MAS and any of them could be used and tailored

to model and implement DM enriched applications. Second, there is a remarkable lack of evaluation methods and tools in the area of MAS. By first outlining the evaluation requirements of our proposed approach, we imply the designing requirements that can be met by some of the existing designing methodologies.

The proposed evaluation methodology serves as an off-the-shelf tool for researchers and developers in this field. Composed of both theoretical analysis and software tools, it provides guidelines and techniques that can be used, adopted or extended for the application domain at hand. We follow the top-down engineering perspective, as proposed by Albus [1] and described in the previous section. The methodology is therefore applicable to existing applications or applications that meet current agent oriented engineering concepts and follow the definitions for agent systems and DM terms provided in previous sections.

In our approach, evaluation derives from the observable behavior of agents within MAS. We therefore consider agents to be black boxes at different levels of granularity, depending on the application scope and the evaluation needs of the designer. In a *fine-grained* level, it is desirable to isolate single agents and observe their efficiency and impact of their actions on their environment. In a *coarse-grained* level, we focus on the overall behavior of an agent society and the outcome of transparent collaborative problem solving, competition or negotiation. In the former case, we consider each participating agent as a black box, whereas in the latter case we view the entire MAS as a black box. In both cases, the methodology isolates and measures certain characteristics of the observable behavior of each black box.

By this approach, we assess the impact of the actual performance of agents and MAS, bypassing the methods of implementation, algorithms and other intrinsic characteristics. This implementation independence of our methodology, makes it a powerful tool for measuring both directly the actual efficacy of a system as a whole and indirectly the performance of the intrinsic methods employed. In other words, the algorithmic decisions are indirectly handled and revised in an iterative manner, if and only if the results of the observable behavior evaluation are not within accepted limits. Moreover, two differently implemented systems can be compared against each other, solely in terms of efficiency, having the underlying mechanisms implicitly compared at the same time.

For establishing an evaluation framework that meets the above characteristics, we define:

- *Horizontal aspects*, the essential methodological steps, that if followed sequentially in an iterative manner, will comprise a complete evaluation methodology. The horizontal aspects of our methodology are:
  - *Definitions and theoretical background* on evaluation terms and relevant techniques.

  - *Theoretical tools* that can help designers chose what to measure, how to measure and how to integrate specific findings.
  - *Software tools* that assist designers in their experimental measurements.
- *Vertical aspects* are specific techniques that may be part of any of the above horizontal aspects and deal with the following three terms[40]:
  - *Metrics* that correspond to system features to be measured.
  - *Measurement methods* that define the actual experimental procedure of assigning measurement values to the selected metrics.
  - *Aggregation* of the metric-measurement pairs in single characterizations for the system.

In the remainder of this section, we examine the above mentioned horizontal aspects in turn, analyzing each of their vertical aspects accordingly.

4.1 Definitions and theoretical background

The definitions of relevant terms and the corresponding theoretical background is of vital importance, in order to determine the scope and goals of evaluation. Any developer, before actually initiating his/her experiments, must have full grasp of what can and what cannot be evaluated. We, hereinafter, present relevant definitions and background theory with respect to: a) metrics, b) measurement methods, and c) aggregation.

Metrics

Metrics are standards that define measurable attributes of entities, their units and their scopes. Metrics are the essential building blocks of any evaluation process, since they allow the establishment of specific goals for improvement. A specific metric provides an indication of the degree to which a specific system attribute has met its defined goal. Deviation from the desired range of values indicates that improvement is needed in the related parts or modules of the system. With respect to a complete evaluation methodology, a metric is the answer to the question: "*What should I evaluate?*".

It must be noted that in software engineering (SE) the term metric is often used interchangeably with the term measurement to denote specific measured values for an attribute. In this work, however, we distinguish the two terms, in order to separate the metrics selection process from the actual measurement data collection.

A metric is defined by: a) its relationship to the corresponding features of the evaluated system, and b) the selected scale of measurement. The former consists of actual parameters or attributes of the system. For example, in auction environments, typical attributes would be the starting and ending time of an auction, as well as the winning bid. The latter term refers to the unit of measurement for this attribute. In the above example, a timestamp

in milliseconds or an amount in Euros would suffice to describe the scale of the selected attributes. Typical types of scales include nominal and ordinal values, intervals and ratio scales.

In SE more than 300 metrics have been defined and used in various evaluation methodologies. Metrics are organized in categories, including technical, performance, business, productivity and end-user metrics[11]. However, in this work, we focus only on performance metrics, since DM for MAS is a relatively new technology that has not yet reached the maturity of software products. The proposed evaluation methodology, however, is expandable so that other aspects besides performance may be covered as well.

A requirement of our methodology with respect to metrics is to be able to provide a set of appropriate metrics and a comprehensive organization of this set in terms of similarity and cohesion. This taxonomy of metrics should also be extensible so that it is applicable to any application. A user will ideally be able to parse this taxonomy and select the metrics that are desirable for the system at hand.

Measurement

Measurement is defined as "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"[32]. 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 "*How should I perform the experimental evaluation?*".

Among the three distinct steps of the proposed methodology, measurement is the most dependent on the implementation details of the application. Indeed, a generalized evaluation methodology cannot indicate precise steps of carrying out a specific measurement procedure. From this point of view, the key requirement of our methodology is to provide a set of measurement methods, so that the user may choose from a variety of options the most appropriate on to tackle the application domain at hand. Our methodology also provides a guideline list for conducting experimental design and data collection, as derived from traditional SE evaluation paradigms.

Aggregation

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). An aggregation process must weightedly balance contradicting measures and provide an overall view of parts or the whole of the system, within boundaries of acceptable performance. Aggregation is the answer to the question: "*What is the outcome of the evaluation procedure?*".

A plethora of diverse aggregation techniques exist. Most commonly, aggregation is accomplished with the assistance of mathematical representation of multi-dimensional vectors and methods of either comparing sets of such vectors or comparing single vectors against predefined ideal vectors. The field of multi-criteria decision making provides us with a rich literature on this issue. Approaches in this field vary from simple sum, average, triangulation or other weighted functions to multi-dimensional vector comparison and vector distance. Especially in the field of intelligent system evaluation, vectors play a crucial role, in the form of Vector of Performance (VoP) or Vectors of Intelligence (VoI) [43].

A requirement of our methodology with respect to aggregation is to provide with the user with appropriate aggregation theory and techniques in order to combine any specific measurements into conclusive characterizations of the evaluated system.

4.2 Theoretical Tools

We next present a set of theoretical tools that aim to assist users throughout the designing of the evaluation procedure, by providing sets of options and guidelines for intelligent performance assessment.

Metrics

Motivated by the requirements for metrics presented above, we introduce a theoretical tool for metric categorization in the form of an acyclic directed graph. The graph is organized in layers or *views* of granularity from general to specific, as further explained below. A user may traverse the graph in a top-down manner and, depending on the choices made, he/she shall conclude to a set of suitable metrics. After finalizing the measurement and aggregation methodology, as presented in the following sections, the developer will be able to traverse the graph upwards in order to provide single characterization for the system at each view. The graph is designed to be general, but also provides the option of extensibility for necessary domain specific metrics.

This tool was inspired by numerous related efforts in the traditional SE field, that provide comprehensive taxonomies of metrics (e.g. [8]. Besides dealing with predictable, deterministic, closed and non-dynamic software, the above approaches also come short because of the flat categorization of metrics at a single level of granularity.

In the proposed approach, we organize a metrics graph into four views, as depicted in Figure 3:

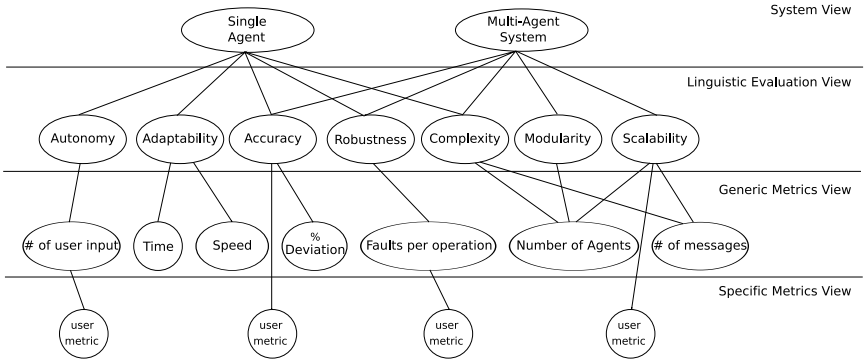

Fig. 3. Metrics graph

These views include:

1. *System view*: At the top-most level, the class of the application is selected. A user may chose between single-agent, multi-agent society and multi-agent competition, depending on the scope and focus of the evaluation effort.
2. *Linguistic evaluation view*: At this level, a user chooses the appropriate verbal characterizations of system aspects, such as accuracy, timeliness, robustness and scalability. These abstract high level characterizations exclude parts of the underlying metrics, while focusing on the aspects of interest to the evaluator.
3. *Generic metrics view*: This level consists of metrics that are general and independent of the application field, such as response time, number of agents and message exchange frequency. The user may either use directly these metrics or refine them by continuing to the next level.
4. *Specific metrics view*: The final level consists of metrics that are specific to the application field. These metrics are only defined by the user, since they are not known *a priori* to a generalized evaluation methodology. Newly defined metrics must conform to the metric definition and parametrization presented in the previous section. Finally, they must be appended to one of the graph nodes of the above levels with directed arcs.

After selecting the metrics from this graph, the user is requested to define a set of parameters for each metric, including the preferred scale of measurement and other attributes, such as frequency of measurement, time intervals etc.

Measurement methods

Before implementing the actual measurement process, one must define the measurement method. Kitchenham [28] provides a categorization of measure-

ment techniques, with respect to the types of properties employed and the nature of the experimental technique. Inspired by this work, we provide the categorization displayed in the Table 2.

Table 2. Categorization of measurement methods

Experiment Type	Description
Quantitative experiment	An investigation of the quantitative impact of methods/tools organized as a formal experiment
Quantitative case study	An investigation of the quantitative impact of methods/tools organized as a case study
Quantitative survey	An investigation of the quantitative impact of methods/tools organized as a survey
Qualitative screening	A feature-based evaluation done by a single individual who not only determines the features to be assessed and their rating scale but also does the assessment. For initial screening, the evaluations are usually based on literature describing the software method/tools rather than actual use of the methods/tools
Qualitative experiment	A feature-based evaluation done by a group of potential user who are expected to try out the methods/tools on typical tasks before making their evaluations
Qualitative case study	A feature-based evaluation performed by someone who has used the method/tool on a real project
Qualitative survey	A feature-based evaluation done by people who have had experience of using the method/tool, or have studied the method/tool. The difference between a survey and an experiment is that participation in a survey is at the discretion of the subject
Qualitative effects analysis	A subjective assessment of the quantitative effect of methods and tools, based on expert opinion
Benchmarking	A process of running a number of standard tests using alternative tools/methods (usually tools) and assessing the relative performance of the tools against those tests

Having selected the measurement method, one must thoroughly provide an experimental design prototype and a data collection procedure. As stated earlier, our methodology can only provide a set of guidelines that any designer may adjust to their specific application. A typical experimental design procedure must describe thoroughly the objectives of the experiments and ensure that these objectives can be reached using the specified techniques.

The last step of the measurement methodology is to carry out the data collection process. Here, the basic guidelines for the designer to follow are to ensure that the data collection process is well defined and monitor the data collection and watch for deviations from the experiment design.

Aggregation

Following the collection of measurement values and the construction of metric-measurement pairs, the problem of aggregation arises. In the evaluation process, aggregation occurs naturally in order to summarize the experimental findings into a single characterization of the performance, either of single modules, or the system as a whole. In the case of the metrics graph of the proposed methodology, after having the measurements collected, the user must traverse the graph in a bottom-up manner. From the *specific metrics view* and the *general metrics view*, he/she must proceed upwards and, at each view, apply aggregation techniques to provide single characterizations for every parent node.

For example, assume that the simplified subtree, depicted in Figure 4, has been used for measuring the performance of a single agent. It must be noted that in this example, no user specified metrics have been included and therefore the *Specific Metrics View* layer is omitted.

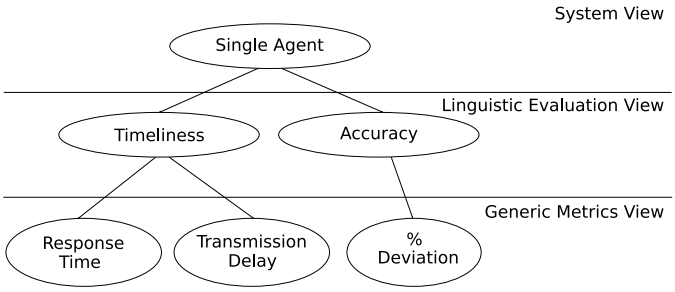

Fig. 4. Example metrics graph for aggregation

We assume that we already have obtained measurement values for the metrics at the general metrics view. In this example, the aggregation process consists of two steps. The first step concludes to a single measurement or characterization for each of the two selected linguistic terms, namely timeliness and accuracy, by weightedly combining the available measurements. The second step combines the linguistic characterization into a single characterization for the single agent system.

It becomes apparent from the above example, that a natural method for combining diverse and heterogeneous measurement information and linguistic characterizations is needed. We argue that *fuzzy aggregation* provides us

with the appropriate natural functionality for this purpose. The term *natural* refers to the ability of the evaluator to express the evaluation findings in a manner that is coherent to their natural language. In other words, the fuzzy aggregation process translates the problem of combining numerical, ordinal or other measures into a collection of verbal characterizations for the system performance.

The proposed fuzzy aggregation method consists of four steps:

1. *Define weights in the metrics graph.* This process determines the importance of each node in the metrics graph with respect to the overall system performance. This decision relies heavily on the application domain as well as the requirements of each application. Hence, the determination of the weights may occur either a) semi-automatically, in case historical data on the importance of each node are available, possibly by an expert system, or b) directly by an expert user, the system designers in most cases.
2. *Define corresponding fuzzy scales for each metric.* The next step deals with the definition of fuzzy scales for the selected metrics. Fuzzy scales are defined by ordinal linguistic variables, such as *low*, *moderate*, *high* and membership functions that map numerical values to the above variables. Having the scales defined, one may already have scales for *natural* characterizations of performance, such as *high response time* or *moderate accuracy*, with respect to desired values.
3. *Convert actual measurements to fuzzy scales.* The conversion is a simple import of the selected measurements to the membership functions defined in the previous step.
4. *Apply a corresponding fuzzy aggregation operator at each view of the graph.* A wide variety of fuzzy aggregation operators exists [19], which can be categorized in:
   - Conjunctive operators, that perform aggregation with the logical "and" connection.
   - Disjunctive operators, that perform aggregation with the logical "or" connection.
   - Compensative operators, which are comprised between minimum and maximum, such as mean or median operators.
   - Non-compensative operators, that do not belong to any of the above categories, such as symmetric sums.

Theoretical tools: Summary

In Table 3, we summarize the required methodological steps with respect to the theoretical tools, which take place at the evaluation process of a development methodology. In section 5.2, we present a real world case study on which the presented methodology is thoroughly applied.

Table 3. Summarization of methodological steps

1. Traverse metrics graph 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 scales and convert measurements accordingly
8. Select and apply aggregation operators on the collected measurements

4.3 Software Tools

In addition to the above presented theoretical tools, a complete off-the-shelf evaluation methodology must contain specifically implemented software tools that assist the evaluation procedure. In this section, we sketch the outline of such a software evaluation tool and the corresponding design guidelines that can be adopted by other similar tools.

The proposed evaluator tool is required to provide semi-automated assistance to the user throughout the following phases of evaluation:

- design
- run-time experimental procedure
- evaluation data summarization
- presentation of the results

A key requirement for this tool is that there must be minimum intervention to the code of an existing system and minimum prior knowledge of the system developer with respect to evaluation aspects. This requirement allows such software tools to apply to existing applications as an evaluation plug-in using dynamic code generation. Figure 5 depicts the six stages of assistance to the evaluation procedure.

1. *Selection of Metrics*: The first component of the proposed tool, is a Graphical User Interface that presents a metrics graph to the user and provides him/her with edit operations. According to the needs of the application, the user selects paths that lead to useful metric nodes and deletes paths and nodes that are unnecessary. As a result, the user determines a subset of the initial graph that is specific to the initiated evaluation procedure. Finally, the parameters of each selected metric as well as corresponding aggregation weights are defined.
2. *Dynamic Interface Generation*: Having the metrics subtree that has been produced in the previous stage, the tool dynamically generates a Java programming interface that corresponds to this evaluation procedure. Elements of agent behavior, communication and interaction are incorporated to implement essential abstract methods that are instantiated in the next stage.

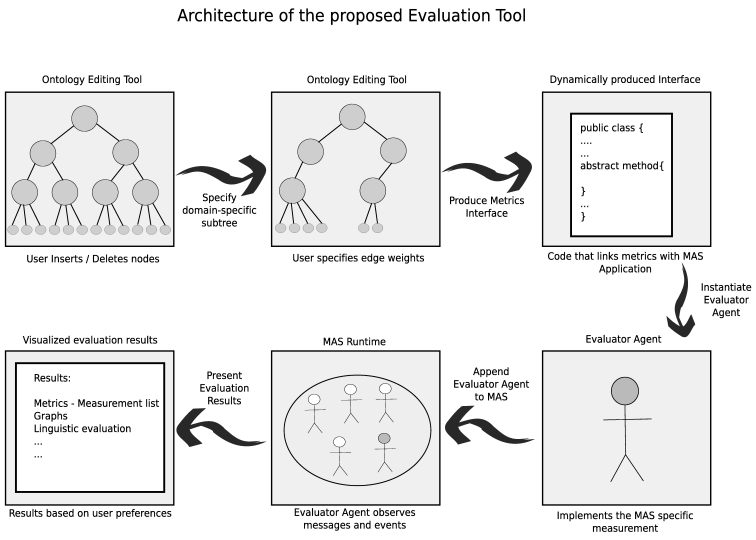

Fig. 5. Outline of the proposed Agent Evaluator Software Tool

3. *Implementation of Evaluator Agent*:Based on the produced programming interface, the user of the tool provide the necessary code for an Evaluator Agent that implements this interface, taking into account all the application-specific details.
4. *Attachment of Evaluator Agent to a MAS at run-time*: The Evaluator Agent is imported to the running system at run-time, executing observation operations on actions, events and messages that relate to the selected metrics.
5. *Data Collection*: The Evaluator Agent records all the observed actions into a log file.
6. *Presentation of Results*: After completion of the system execution or the ellapse of a predefined interval, the proposed tool presents the outcome of the evaluation process to the user, by processing the log files, deriving appropriate measurements from recorded events and messages and aggregates the results accordingly.

5 Agent Academy II: an Agent-Oriented Software Engineering tool for building DM-enriched MAS

5.1 Agent Academy II

Agent Academy is an open-source project, currently in the second version[1]. The design and implementation has changed radically since version 1.0. The initial goals of the application have not changed though, focusing mainly in the integration of agent developing and data mining technologies. We tried also to make this integration as broad as possible. In this perspective, we added features we find valuable to developers such as project management capabilities or java editing tools. Agent Academy enables users to reuse existing code by allowing a user to simply paste it into AA directories.

Agent Academy consists of three basic tools that form also an abstraction of creating a Multi Agent System. These tools are:

- Behavior Design Tool, where the user can create JADE behaviors, compile them and use a number of capabilities that this tool provides (for example users can create java source code that sends/receives ACL messages using a simple GUI etc) A central feature of the BDT is that it supplies to the users an intuitive tool for keeping the source code structured. The idea behind this feature is that users can create blocks of code of any size (i.e. it can span in an entire method). These blocks have a description, a body (the source code) and the method in which they belong. All these attributes can change dynamically allowing the users to add the same block of code in many methods or create a high-level description of the source file by summing up all the blocks descriptions. Implementing these blocks as static variables enables users to exchange blocks among multiple source files.
- Agent Design Tool, with which the developers can create software agents and add the designed behaviors as the agents execution tasks. We think agent functionality should reside in the agents behaviors, so ADT has not extended capabilities. ADT is equipped with a number of capabilities that we think are really important to agent developers. We are going to briefly mention them here and describe them in greater detail later. The first among them is the debugging tool (aka Graphical Monitor Agent Execution) that its use can save lot of debugging time. By means of this monitor, users can watch the execution of agent behaviors. This process is dynamic, meaning that behaviors that are added to the agent long after agent initialization can also be monitored. A tool that is also available from other Agent Academy modules is the Data Mining Module. Users can launch this tool, create a data mining model in the fly and add it to agent code just like any other JADE behavior.

[1] Available at http://sourceforge.net/projects/agentacademy

- Multi-Agent System Design Tool, is the tool that developers can use to create Multi-Agent Systems. The functionality of this tool is limited, constrained mainly in the easy, user-friendly definition of the participants in the MAS being designed.

Agent Academy users are encouraged to implement the MAS creation process following steps that correspond to the order the AA tools were presented, although other approaches can also be effective. Beyond this agent-oriented functionality, Agent Academy comes with a set of tools for a small/medium scale Project Management. In more detail, users are given the possibility to keep a small account of their projects components by using the Project Notepad that is matrix in which they can store the agents and behaviors created inside their project. Users can also use standard clean/build/run project capabilities from the Agent Academy main window.

5.2 A real world demonstrator

For validating the proposed methodology, we have selected Supply Chain Management (SCM) as a representative domain for testing agents that utilize DM techniques. We have implemented an SCM agent under the name Mertacor that has successfully participated in past Trading Agent SCM Competitions. Mertacor combines agent features with DM techniques. In the remainder of this section, we provide an overview of the SCM domain, the competition scenario and Mertacor's architecture. We conclude by applying the proposed evaluation methodology to different implementations of Mertacor.

Supply Chain Management

SCM tasks comprise the management of materials, information and finance in a network consisting of suppliers, manufacturers, distributors and customers. SCM strategies target at the efficient orchestration of the sequence of tasks, from raw materials to end-user service. Traditional SCM relied heavily on rigid and predefined contracts between participating parties. However, the need for dynamic configuration of the supply chain, as indicated nowaydays by global markets, became imperative. Modern SCM approaches focus on the integration, optimization and management of the entire process of material sourcing, production, inventory management and distribution to customers.

The core design primitives for coping with SCM are a) coordination of distributed, heterogeneous information,b) efficient negotiation between participating entities and c) functional resource allocation. MAS are an ideal modelling and implementation solution to this inherently distributed problem [12][49]. Moreover, DM techniques have been successfully applied for SCM purposes in the past. DM has efficiently addresses issues of customer and supplier profiling, inventory scheduling and market based analysis.

SCM Trading Agent Competition Game

The Trading Agent Competition (TAC) is an annual, international competition that consists of two games: a) TAC Classic and b) TAC SCM. In the latter, the game scenario, as described in [7], consists of groups of six competing agents, each of which represents a PC assembler with limited production capacity an competes with other agents in selling PC units to customers. The agents' task is to efficiently manage a part of the supply chain, namely to negotiate on supply contracts, bid for customer offers, manage daily assembly activities and ship completed offers to customers. Negotiations with manufacturers and customers are performed through a Request-For-Quote (RFQ) mechanism, which proceeds in three steps:

- Buyer issues RFQs to one or more sellers
- Sellers respond to RFQs with offers
- Buyers accept or reject offers. An accepted offer becomes an order.

In order to get paid, the agent must deliver on-time, otherwise it is charged with a penalty. At the end of the game, the agent with the greatest revenue is declared winner. For more information on the game, read the game specification provided by [7].

Mertacor Architecture

Mertacor, as introduced in [31], consists of four cooperating modules (Figure 6):

1. the *Inventory Module*(IM). Mertacor introduces an assemble-to-order (ATO) strategy, which is a combination of two popular inventory strategies, namely *make-to-order* and *make-to-stock*.
2. the *Procuring Module*(PM). This module predicts future demand and orders affordable components, balancing between cheap procurement and running needs in the assembly line.
3. the *Factory Module*(FM). This module constructs assembly schedules and provides the Bidding Module with information on the factory production capacity, based on simulation of customer demand for the next 15 game days.
4. the *Bidding Module*(BM). This module attempts to predict a winning bid for each order, by performing DM on logs of past games, and makes respective offers for incoming orders.

Mertacor's core integrates these modules into a transparently robust unit that handles negotiations with both customers and suppliers. This architecture provides flexibility and extensibility, permitting the application of Mertacor's strategy to other real-life SCM environments.

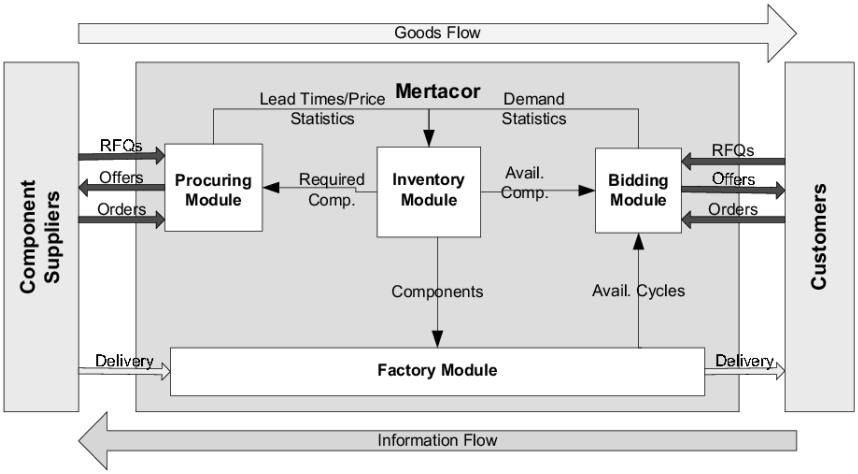

Fig. 6. Overview of Mertacor's architecture

Evaluating Mertacor's performance

In the remainder of this section, we apply the proposed evaluation methodology to various implementations of Mertacor. In our effort to assess the impact of DM in Mertacor's performance, we require that the experiments are planned is such way that deals with both DM algorithmic-specific efficacy and their impact on overall agent performance. We follow the methodological steps defined in Table 3.

Step 1: Traverse metrics graph and select metrics

Starting from the *System view*, we select the *Single Agent* node and its corresponding path. This choice is attributed to the nature of auctioning environments; we, being the developers of Mertacor, have complete control only on the agent's executing thread and observe the auctioning world only through Mertacor's perspective. We, therefore, need to focus on performance aspects that exclusively deal with this single agent.

At the *Linguistic Evaluation View*, we select the linguistic metrics of *Accuracy*, *Timeliness* and *Adaptability*. Indeed, from our experience in SCM auctions, these three characteristics are the most significant ones, since the outcome of each auction is heavily dependent on the deviation of the forecasted bid, the on-time delivery of the bid and the ability of the agent to adapt in dynamic environments, respectively.

At the *Generic Metrics View* we only select *Time*, as the standard metric for *Timeliness*. The rest of the metrics are domain specific and are, therefore, defined in the next methodological step.

Step 2: Provide domain specific metrics

Metrics for *Accuracy* should directly refer to DM related performance, since the outcome of the application of DM is directly related to the selected bid. For this purpose, we have selected the *Correlation Coefficient* (cc), the *Relative Absolute Error* (RAE) and the *Root Mean Square Error* (RMSE) metrics.

Finally, for *Adaptability*, we have selected *Competition Intensity* that defines the participation intensity in the auction environment. In highly competitive environments, our agent is required to exhibit a larger degree of adaptability.

An instance of the metrics graph for this evaluation effort is depicted in Figure 7.

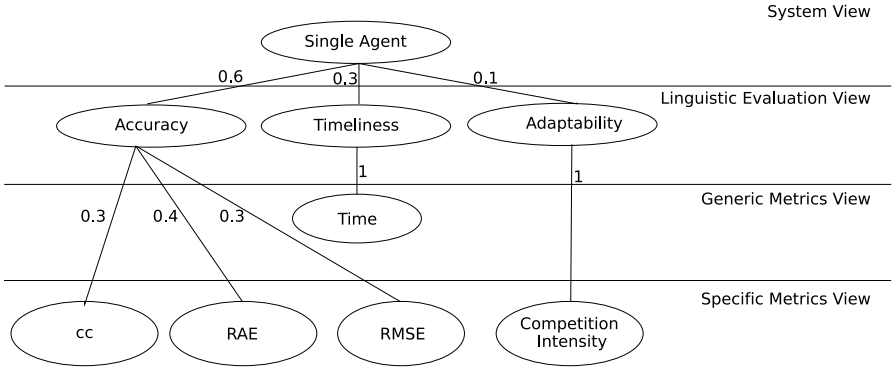

Fig. 7. Resulted metrics graph for Mertacor evaluation

Step 3: Determine metrics parameters

We now continue by defining the scale of each metric. For the three linguistic metrics, *Accuracy*, *Timeliness* and *Adaptability*, we define the corresponding fuzzy scales in *Step 7* of the methodology. For the generic and specific metrics, we provide the following scales:

1. *CC*: The correlation coefficient is the degree at which the forecasted bid and the resulted price are correlated. The *cc* lies in the [-1,1] interval.
2. *RAE*: The *Relative Absolute Error* is a percentage indicator for the deviation of the above mentioned variables.
3. *RMSE*: The *Root Mean Square Error* is another well-known DM metric for the above mentioned variables.
4. *Time*: In TAC SCM auctions, bids are normally submitted just before the end of each predefined auction interval. One could argue that, since this time constraint exists, all agents have a time barrier to bid and therefore

all bidding calculation procedures should be characterized either as successful or failed. In that context, timeliness is only a binary metric that provides no further performance indication. However, due to the modular architecture of Mertacor, the earliest possible decision on the bid, allows the agent to perform other game-related tasks in this interval. We therefore define *Time* as the time interval between the first call of the related bidding API function and the determination of the bidding value, in milliseconds.

5. *Competition Intensity*: We have selected two different competing environments that affect *Adaptability*: a) Finals, where the competition intensity is high, and b) Second finals, where the competition intensity is low.

Step 4: Specify measurement method and parameters

Estimation of the winning price of the bids can be modeled as a regression problem, where the desired output is the agent's bidding price for clients' RFQs and the inputs are the parameters related to the bid that are known to the agent. The initial set of attributes considered are the demand (Total PCs requested each day), the demand in the product's market range, the due date of the order, the reserve price of components, and the maximum and minimum prices of same type PCs sold in the last days (2 previous days for maximum 4 for minimum), as shown in Table 4.

Table 4. Set of SCM auction attributes for DM

Attribute description	Attribute name
Demand (Total PCs requested the day the RFQ was issued)	demandAll
Demand in the product's market range	demandRange
Duedate	dueDate
Reserve price	reservePrice
Maximum price of PCs of same type sold in the last 1 day	max1
Maximum price of PCs of same type sold in the last 2 days	max2
Minimum price of PCs of same type sold in the last 1 day	min1
Minimum price of PCs of same type sold in the last 2 days	min2
Minimum price of PCs of same type sold in the last 3 days	min3
Minimum price of PCs of same type sold in the last 4 days	min4
Winning price of the bid	price

Available data was split into three subsets, each one representing a different market range (LOW - MEDIUM - HIGH), both for the finals and second finals of the game, resulting to six different datasets (finalsLOWMEDIUMHIGH and secondFinalsLOWMEDIUMHIGH).

The instances within the initial datasets ranged from 45000 to 230000 instances. Analysis was performed on all datasets. Nevertheless, due to space limitation we shall discuss only one case (finalsLOW dataset), while analysis

on the other cases was performed in an analogous manner. The initial dataset contained 156228 records of bids. In order to remove redundant information and enable quicker and more accurate training, a number of pre-processing filters were tested against the dataset. In particular, we applied the CfsSubsetEval1 [21] WrapperSubsetEval2 [30], and ReliefFAttributeEva3l [41] filters for attribute selection, using the GreedyStepwise and RandomSearch search methods. The trimmed dataset contained the following attributes as input: the demand, the reserve price for components, the maximum price of same type PCs for the two previous days and the minimum price for the previous day, while price was the output attribute. In order to reduce the number of instances for training, and since the class attribute (price) is numeric, the StratifiedRemoveFolds4 [5] method was selected. Two datasets were finally produced, containing the one third (1/3) and one eighth (1/8) of the initial instances respectively.

Finally, for training purposes, four different classification (regression) and two meta-classification schemes were applied, in order to decide on the one that optimally meets the problem of predicting the winning bid of an order:

1. Linear Regression
2. Neural Networks
3. SMOreg (Support Vector Machines)
4. the M5' algorithm
5. Additive Regression
6. Bagging

Step 5: Execute experiments

In order to experiment on the data with a variety of training techniques and algorithms, the WEKA [47] was selected, providing with a wide range of filters for pre-processing, model evaluation, visualization and post-processing. The results of the experimental procedures are presented in Table 5 and Table 6, for low and high *Competition Intensity* values, respectively.

Step 6: Define weights in the graph

This step requires a subjective, expert-initiated attribution of weights to the corresponding edges of the metrics graph. Driven by our experience in the field, we assign a higher weight to *Accuracy* (0.6) and lesser weights to *Timeliness* (0.3) and *Adaptability* (0.1). The corresponding weights are illustrated in Figure 7.

Step 7: Define fuzzy scales and convert measurements accordingly

We provide the following fuzzy sets for the selected metrics:

- Fuzzy variables *very low*,*low*,*medium*,*high* and *very high* for the *RAE* and *RMSE* metrics

Table 5. Results of experiments for low Competition Intensity

Algorithm	CC	RAE (%)	RMSE	Time	Data Subset
Linear Regression	0.93	28.99	90.17	108	LOW
Neural Networks	0.93	32.91	94.69	111	LOW
Support Vector Machines	0.93	26.47	89.08	157	LOW
M5'	0.95	22.77	61.09	140	LOW
Additive Regr.	1.00	3.21	22.12	192	LOW
Bagging	0.98	14.89	52.02	201	LOW
Linear Regression	0.94	26.50	112.71	129	MEDIUM
Neural Networks	0.95	24.85	105.69	133	MEDIUM
Support Vector Machines	0.93	28.66	109.61	182	MEDIUM
M5'	0.97	19.30	86.90	168	MEDIUM
Additive Regr.	1.00	3.01	23.53	230	MEDIUM
Bagging	0.98	13.26	65.52	237	MEDIUM
Linear Regression	0.94	26.78	105.51	140	HIGH
Neural Networks	0.95	27.83	105.21	144	HIGH
Support Vector Machines	0.94	25.82	103.32	204	HIGH
M5'	0.96	21.29	87.14	183	HIGH
Additive Regr	1.00	2.98	24.70	249	HIGH
Bagging	0.98	15.13	65.69	260	HIGH

Table 6. Results of experiments for high Competition Intensity

Algorithm	CC	RAE (%)	RMSE	Time	Data Subset
Linear Regression	0.98	14.34	63.40	110	LOW
Neural Networks	0.97	21.26	64.82	112	LOW
Support Vector Machines	0.96	17.48	72.84	155	LOW
M5'	0.98	13.49	56.79	145	LOW
Additive Regr.	0.97	19.24	67.51	189	LOW
Bagging	0.99	5.62	27.76	199	LOW
Linear Regression	0.97	16.95	74.33	133	MEDIUM
Neural Networks	0.97	20.21	75.20	132	MEDIUM
Support Vector Machines	0.97	17.54	73.81	193	MEDIUM
M5'	0.99	9.91	46.93	172	MEDIUM
Additive Regr.	0.96	25.98	92.30	227	MEDIUM
Bagging	1.00	4.84	31.38	233	MEDIUM
Linear Regression	0.97	16.55	68.14	142	HIGH
Neural Networks	0.98	18.94	71.91	144	HIGH
Support Vector Machines	0.97	16.27	72.31	208	HIGH
M5'	0.99	10.03	45.35	178	HIGH
Additive Regr.	0.95	28.26	94.68	242	HIGH
Bagging	0.99	5.70	34.90	263	HIGH

- Fuzzy variables *low* and *high* for the *CC* and *Competition Intensity* metrics
- Fuzzy variables *low*, *medium* and *high* for the *Time* metric

The corresponding fuzzy membership functions for *CC*, *RAE*, *RMSE* and *Time* are depicted in Figure 8.

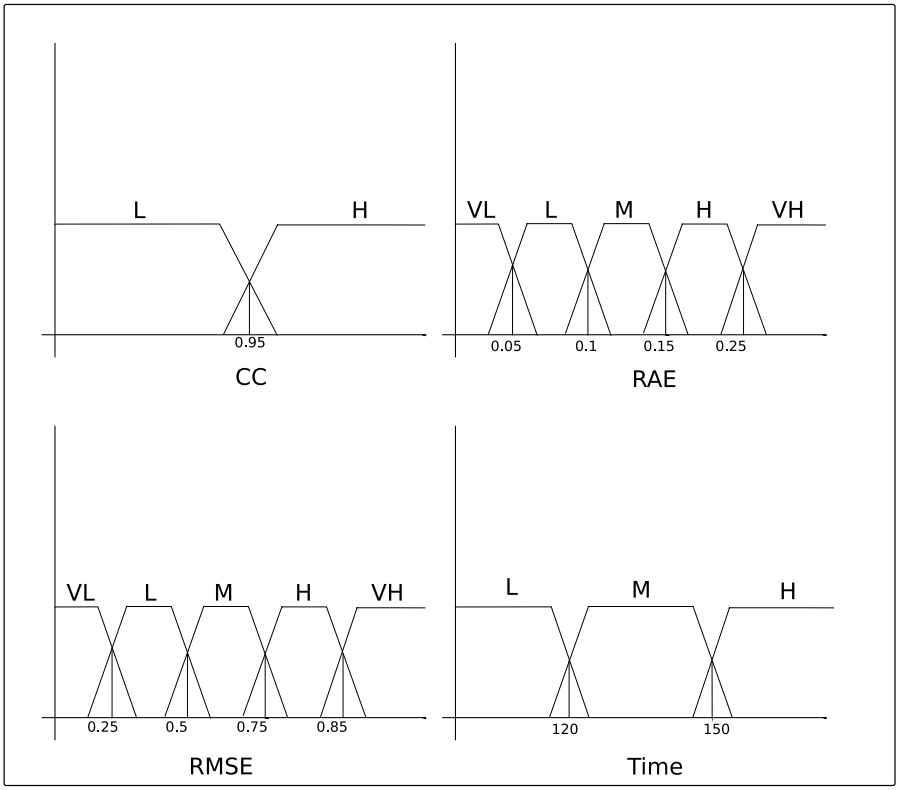

Fig. 8. Fuzzy membership functions for the selected metrics

Step 8: Select and apply aggregation operators on the collected measurements

The final step of the methodology consists of the application of the selected aggregation method. As described in [19], the application of weighted operators result into a single characterization for every linguistic metric. After summarizing the results it can be seen that *Additive Regression* exhibit the best performance for all data subsets, as it balances between large accuracy and adequate time responses, for both high and low *Competition Intensity*.

6 Concluding remarks

As the number of application that integrate DM and AT increase, the need for assessing the overall system performance is imperative. In this work, we have presented a generalized methodology for evaluating agents and MAS that employ DM techniques for knowledge extraction and knowledge model generation. The proposed methodology comprises a set of concise steps that guide an evaluator through the evaluation process. A novel theoretical representation tool introduces a metrics graph and appropriate selection guidelines for measurement and aggregation methods. A real world DM-enriched agent in the field of Supply Chain Management has used to demonstrate the applicability of the proposed methodology. Future work in this direction include the specification of a unique metrics ontology for the proposed metrics representation graph and the expansion of the graph with a complete set of real world metrics, borrowed either from the software engineering discipline or existing, ad-hoc efforts in intelligent systems evaluation. Finally, the proposed methodology must be thoroughly tested in a number of diverse and representative case studies.

Acknowledgement

This paper is part of the 03ED735 research project, implemented within the framework of the Reinforcement Programme of Human Research Manpower (PENED) and cofinanced 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] James Albus, Elena R. Messina, and John M. Evans. Performance metrics for intelliget systems (permis) white paper. In *Proc. of the First International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 12-14 August 2000.

[2] Dennis Allard and Chris Fraley. Non parametric maximum likelihood estimation of features in saptial point process using voronoi tessellation. *Journal of the American Statistical Association*, 1997.

[3] Brian W. Arthur. Inductive reasoning and bounded rationality. *American Economic Review*, 84:406–411, 1994.

[4] Ioannis N. Athanasiadis and Pericles A. Mitkas. Social influence and water conservation: An agent-based approach. *Computing in Science and Engg.*, 7(1):65–70, 2005. ISSN 1521-9615. doi: http://dx.doi.org/10.1109/MCSE.2005.21.

[5] Friedman J.H. unsrt Olshen R.A. Breiman, L. and C.J. Stone. *Classification and Regression Trees*. Chapman and Hall, New York, 1984.

[6] J. L. Burke, R.R. Murphy, D. R. Riddle, and T. Fincannon. Task performance metrics in human-robot interaction: Taking a systems approach. In *Proc. of the Fourth International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 13-15 August 2002.

[7] Arunachalam R. Sadeh N. Ericsson J. Finne N. Collins, J. and S. Janson. The supply chain management game for the 2005 trading agent competition. Technical report, CMU, 2004.

[8] M. de los Angeles Martin and Luis Olsina. Towards an ontology for software metrics and indicators as the foundation for a cataloging web system. In *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*, page 103, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2058-8.

[9] Christos Dimou, Alexandros Batzios, Andreas L. Symeonidis, and Pericles A. Mitkas. A multi-agent simulation framework for spiders traversing the semantic web. In *Web Intelligence*, pages 736–739. IEEE Computer Society, 2006. ISBN 0-7695-2747-7.

[10] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *KDD*, pages 82–88, 1996.

[11] Norman E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., London, UK, UK, 1991. ISBN 0442313551.

[12] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 0201360489.

[13] A. A. A. Fernandes. Combining inductive and deductive inference in knowledge management tasks. In *DEXA '00: Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, page 1109, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0680-1.

[14] Ian T. Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *CoRR*, cs.AR/0103025, 2001.

[15] Ian T. Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn: Why grid and agents need each other. In *AAMAS*, pages 8–15, 2004.

[16] Boris Galitsky and Rajesh Pampapathi. Deductive and inductive reasoning for processing the claims of unsatisfied customers. In Paul W. H. Chung, Chris J. Hinde, and Moonis Ali, editors, *IEA/AIE*, volume 2718 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2003. ISBN 3-540-40455-4.

[17] Nicholas Gibbins, Stephen Harris, and Nigel Shadbolt. Agent-based semantic web services. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 710–717, New

York, NY, USA, 2003. ACM Press. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775251.

[18] M.A. Goodrich, E.R. Boer, J. W. Crandall, R.W. Ricks, and M.L. Quigley. Behavioral entropy in human-robot interaction. In *Proc. of the Fourth International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 13-15 August 2002.

[19] Michel Grabisch, Sergei A. Orlovski, and Ronald R. Yager. Fuzzy aggregation of numerical preferences. pages 31–68, 1998.

[20] Amy R. Greenwald and Peter Stone. Autonomous bidding agents in the trading agent competition. *IEEE Internet Computing*, 5(2):52–, 2001.

[21] M. A Hall. Correlation-based feature subset selection for machine learning. Technical report, Thesis submitted in partial fulfilment of the requirements of the degree of Doctor of Philosophy at the University of Waikato, 1998.

[22] X. Hu and B. Zeigler. Measuring cooperative robotic systems using simulation-based virtual environment. In *Proc. of the Fourth International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 13-15 August 2002.

[23] Markus C. Huebscher and Julie A. McCann. Evaluation issues in autonomic computing. In *Proceedings of Grid and Cooperative Computing Workshops (GCC)*, pages 597–608, 2004.

[24] Bernard J. Jansen, Tracy Mullen, Amanda Spink, and Jan Pedersen. Automated gathering of web information: An in-depth examination of agents interacting with search engines. *ACM Trans. Inter. Tech.*, 6(4):442–464, 2006. ISSN 1533-5399. doi: http://doi.acm.org/10.1145/1183463.1183468.

[25] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/367211.367250.

[26] Nicholas R. Jennings. Agent-oriented software engineering. In Ibrahim F. Imam, Yves Kodratoff, Ayman El-Dessouki, and Moonis Ali, editors, *IEA/AIE*, volume 1611 of *Lecture Notes in Computer Science*, pages 4–10. Springer, 1999. ISBN 3-540-66076-3.

[27] Bob Kero, Lucian Russell, Shalom Tsur, and Wei-Min Shen. An overview of database mining techniques. In *KDOOD/TDOOD*, pages 1–8, 1995.

[28] Barbara Ann Kitchenham. Evaluating software engineering methods and tool, part 2: selecting an appropriate evaluation method technical criteria. *SIGSOFT Softw. Eng. Notes*, 21(2):11–15, 1996. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/227531.227533.

[29] Y. Kodratoff. *Introduction to machine learning.* Pitman Publishing, 1988.

[30] R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial Intelligence journal, special issue on relevance*, 97(1-2):273–324, 1997.

[31] I. Kontogounis, Chatzidimitriou, A. K., Symeonidis, and P.A. Mitkas. A robust agent design for dynamic scm environments. In *LNAI, Vol. 3955*, pages 127–136. Springer-Verlag, 2006.

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

[33] Ingo Muller, Ryszard Kowalczyk, and Peter Braun. Towards agent-based coalition formation for service composition. In *IAT '06: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2006 Main Conference Proceedings) (IAT'06)*, pages 73–80, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2748-5. doi: http://dx.doi.org/10.1109/IAT.2006.122.

[34] A. Negri, A. Poggi, M. Tomaiuolo, and P. Turci. Agents for e-business applications. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 907–914, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-303-4. doi: http://doi.acm.org/10.1145/1160633.1160795.

[35] A. L. Nelson, E. Grant, and T. C. Henderson. Competitive relative performance evaluation of neural controllers for competitive game playing with teams of real mobile robots. In *Proc. of the Third International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 13-15 August 2002.

[36] Hyacinth S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):1–40, 1996.

[37] Anna Perini, Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Towards an agent oriented approach to software engineering. In Andrea Omicini and Mirko Viroli, editors, *WOA*, pages 74–79. Pitagora Editrice Bologna, 2001. ISBN 88-371-1272-6.

[38] Martin K. Purvis, Stephen Cranefield, Roy Ward, Mariusz Nowostawski, Daniel Carter, and Geoff Bush. A multi-agent system for the integration of distributed environmental information. *Environmental Modelling and Software*, 18(6):565–572, 2003.

[39] J. Scholtz, B. Antonishek, and J. Young. Evaluation of human-robot interaction in the nist reference search and rescue test arenas. In *Proc. of the Fourth International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 13-15 August 2002.

[40] T. K. Shih. Evolution of mobile agents. In *Proc. of the First International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 12-14 August 2000.

[41] M.R. Sikonja and I. Kononenko. An adaptation of relief for attribute estimation on regression. machine learning. In *Proceedings of 14th International Conference on Machine Learning D., Fished (ed.)*, Nashville, TN, USA, 1997.

[42] Andreas L. Symeonidis and Pericles A. Mitkas. *Agent Intelligence Through Data Mining*. Springer Science and Business Media, 2005.

[43] Tadeusz Szuba. Universal formal model of collective intelligence and its iq measure. In *CEEMAS '01: Revised Papers from the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems*, pages 303–312, London, UK, 2002. Springer-Verlag. ISBN 3-540-43370-8.

[44] Jia Tang and Minjie Zhang. An agent-based peer-to-peer grid computing architecture: convergence of grid and peer-to-peer computing. In *ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research*, pages 33–39, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920-68236-8.

[45] Gerhard Weiss. Agent orientation in software engineering. *Knowl. Eng. Rev.*, 16(4):349–373, 2001. ISSN 0269-8889. doi: http://dx.doi.org/10.1017/S026988890100025X.

[46] Michael P. Wellman and Peter R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24(3-4):115–125, 1998.

[47] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005.

[48] Michael Wooldridge. Intelligent agents. In G. Weiss, editor, *Multiagent Systems*. The MIT Press, April 1999.

[49] J. Wu, M. Ulieru, M. Cobzaru, and D. Norrie. Supply chain management systems: state of the art and vision. In *9th International Conference on Management of Innovation and Technology*, pages 759–764. IEEE Press, 2000. ISBN 3-540-43370-8.

[50] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The michigan internet auctionbot: a configurable auction server for human and software agents. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 301–308, New York, NY, USA, 1998. ACM Press. ISBN 0-89791-983-1. doi: http://doi.acm.org/10.1145/280765.280847.

[51] Lotfi 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.

[52] N. Zimmerman, C. Schlenoff, S. Balakirsky, and R. Wray. Performance evaluation of tools and techniques for representing cost-based decision criteria for on-road autonomous navigation. In *Proc. of the Third International Workshop on Performance Metrics for Intelligent Systems (PERMIS)*, 3-15 August 2002.