Agent-Mining of Grid Log-Files: A Case Study

Arjan J.R. Stoter[1], Simon Dalmolen[1,2], and Wico Mulder[1]

[1] Logica, Amstelveen, The Netherlands
{arjan.stoter,simon.dalmolen,wico.mulder}@logica.com
[2] School of Management & Governance,
University of Twente Enschede, The Netherlands

Abstract. Grid monitoring requires analysis of large amounts of log files across multiple domains. An approach is described for automated extraction of job-flow information from large computer grids, using software agents and genetic computation. A prototype was created as a first step towards communities of agents that will collaborate to learn log-file structures and exchange knowledge across organizational domains.

Keywords: Grid monitoring, text mining, agent oriented programming, genetic computation, engineering.

1 Introduction

Grid infrastructures are distributed and dynamic computing environments that are owned and used by a large number of individuals and organizations. The EGEE [1] grid for instance builds on the collective efforts of over 140 organizations in 50 countries, where each organisation owns and manages part of the grid. In such environments, computational power and resources are shared to process jobs. A job is a computation task launched by a client to be handled by the grid. This job is pointed to a resource by a scheduler and then executed on multiple resources in different parts of the grid, supported by different cluster organizations.

Operational management of grids is challenging. There are many components and interactions, resources may join and leave any time, resources are heterogeneous and distributed across organizations, and the components undergo continuous improvements and changing of standards [2]. The collaborative and often dynamic settings of grids requires an intelligent information management system that is not only flexible and extensible but also able to relate information from different organizational domains.

A rich source of information for managing grids are system log-files. Log files can be used for a number of things, such as performance analysis, security management, and user profiling [3]. In grids, log files are used to analyse network paths and job flows.

Today, system managers in grids use a variety of tools to monitor the status of the grid, such as Monalisa, Nagios, BDII, and RGMA. However, these tools

L. Cao et al.: ADMI 2012, LNAI 7607, pp. 166–177, 2013.

are typically restricted to the boundaries of a single organizational domain. An organisational domain in this case refers to a cluster organisation that owns and maintains a hardware cluster of the grid.

In practice, grid log-files are often analysed manually by local domain administrators. The main reason for this is that the log-files not only contain information about jobs but also other entries related to for instance system performance and security. Job-flow analysis therefore requires specific knowledge about the relevant entries related to job processing and the way in which they are structured. This makes job-flow analysis a complex task for administrators, especially when administrators from different domains have to work together to retrace errors in job processing [2].

Figure 1 illustrates collaboration between grid administrators. Each domain contains log files (manifests) with sometimes different structures. Retrieving the executed path of a job and finding the reason of failure is done manually by domain administrators. Using his or her domain knowledge, administrators scan the manifests for regions of interest (ROIs). An ROI is, for example, an IP address or a unique job identifier. Some administrators make use of regular expression to represent ROIs that match log entries related to jobs. He or she then manually creates an information extraction pattern to extract rich information from the manifests. This information extraction pattern is specific to his or her domain due to the differences in log-file structures across domains. Defining ROIs and domain specific patterns for information extraction is challenging and makes error tracing in grids a complex and time-consuming task, which often requires communication between domain administrators to exchange their domain knowledge to achieve a cross domain overview.

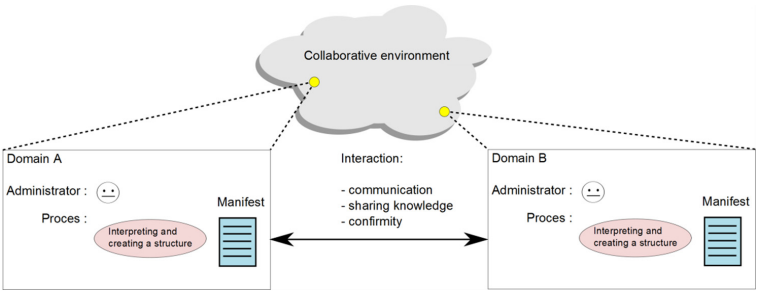

Fig. 1. Schematic representation of collaboration between grid administrators in different domains

2 Approach

Error tracing in grids could be optimized profoundly if administrators would be able to automatically exchange domain knowledge and then automatically incorporate this acquired knowledge into a new and improved pattern for information-extraction from the manifests within their domain.

In the current study, a prototype was created that integrated methodologies from multi-agent technology [4], data mining, and knowledge discovery, known as agent mining [5]. Agent mining has shown a high potential to enhance collaboration performance and tackling of errors and exceptions in distributed computing environments, such as grids and clouds [6].

The current prototype incorporated techniques from agent-based data mining, where agents collaborated inside a multi-agent system (MAS) to optimize information retrieval and gathering of information across distributed nodes in the grid by means of distributed learning [7]. Each agent inside the MAS located and refined ROIs inside log files using domain knowledge and created a domain specific pattern for information extraction, called a corpus. The term corpus stems from text mining, a research area that focuses on the identification and extraction of relevant features inside manifests. In text mining, unstructured or semi-structured data is first transformed into a structured intermediate format. This structured format -or corpus- can then be queried to extract data from manifests [8]. By exchanging their knowledge, the agents learned from one another to achieve higher levels of data abstraction. The goal of the current study was to investigate whether such an approach has the potential to support domain administrators in grids during error tracing.

Figure 2 shows the scope of the prototype. Manifests containing log data from actual grids were used as data. During preprocessing, initial domain knowledge was acquired about the manifests based on a priori knowledge from system administrators. This knowledge consisted of domain knowledge and structure knowledge. Domain knowledge contained a list of attributes that the agent assumed to be of relevance in the log file, the ROIs. These are the data patterns the agent would search for in the log file. These attributes were called descriptive

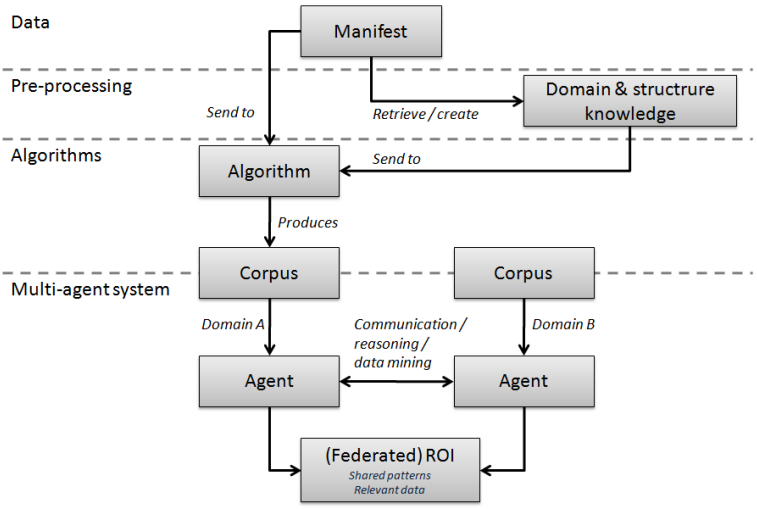

Fig. 2. Scope of the prototype

attributes (DA). DAs were represented by regular expressions, as shown in table 1. Structure knowledge contained a collection of concepts that serve as building blocks to construct the regular expressions.

Table 1. Examples of DAs and ROIs in domain knowledge

DA/ROI	**Log-file line example**
TIME\pPunct*\s*({\w+ \s*:}*)	TIME: Sun Dec 7 04:02:09 2008
PID\pPunct*\s*(\w1,)	PID: 125690

Domain- and structure knowledge were used by an algorithm to locate the ROIs within the manifest, construct a regular expression to represent each ROI, and create a corpus. Finally, the constructed ROIs were federated between the software agents.

Genetic computation was used to construct the corpus. Genetic algorithms use Darwin's principle of natural selection, along with analogs of recombination (crossover), mutation, gene duplication, gene deletion, and mechanisms of developmental biology [9]. A genetic algorithm follows an evolutionary path towards a solution. These solutions are represented by chromosomes made up of individual elements called genes. Genes encode specific kinds of information. Populations of chromosomes (i.e. possible solutions) evolve into new generations of chromosomes by means of recombination (crossover) and mutation. A genetic algorithm typically has the following logic [10]:

1. Create a population of random chromosomes.
2. Test each chromosome for how well it fits the solution.
3. Assign each chromosome a fitness score.
4. Select the chromosomes with the highest fitness scores and allow them to survive to a next generation.
5. Create a new chromosome by using genes from two parent chromosomes (crossover).
6. Mutate some genes in the new chromosome.

Steps 2 to 6 are repeated until a certain condition is met, or when a maximum number of generations is reached.

After construction of the corpus, the agents federated domain knowledge to other agents inside the MAS, who in turn incorporated these new ROI definitions inside their domain knowledge and used them to create a new corpus to read manifests. The JADE framework was used to create a MAS. JADE is an Open-Source, Java-based framework, and one of the most widespread agent-oriented middleware in use today [11].

3 Prototype

Figure 3 shows a schematic representation of the scope of a single agent. Each element of the figure is explained in the next paragraphs.

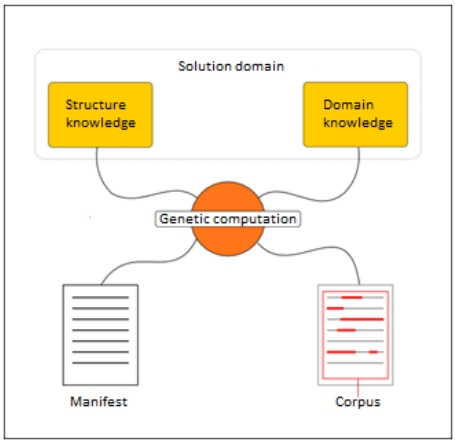

Fig. 3. Representation of the elements within the scope of a single agents

The used data were manifests that contained original log data from the EGEE grid, provided by the National Institute for Nuclear and High energy physics (NIKHEF).

The solution domain in figure 3 represents the agent's pre-knowledge of the manifest: its assumptions about the manifest's structure and interesting parts. The solution domain consists of domain knowledge and structure knowledge. Domain knowledge and structure knowledge together provide the building blocks for an agent to construct a corpus. The corpus was a collection of ROIs ordered in a sequence that has the highest fit on a reoccurring pattern in the manifest.

The solution of genetic computation was a chromosome that represented a corpus that fitted a data pattern inside the manifest. The chromosomes consisted of elements from structure knowledge and domain knowledge, which formed the genes within the chromosome. Each chromosome represented a potential corpus. The size of chromosome was a fixed size, which was set before run-time. The corpus was represented by a directed graph as shown in figure 4. Directed graph G is

$$G = (N, E)$$

where N is the set of nodes and E is the set of edges. The edge set E is a subset of the cross product $(N * N)$. Each element (u,v) in E is an edge joining node u to node v. A node v is neighbour of node u if edge (u,v) is in E.

An $(N * N)$ adjacency matrix (table 2) represented the adjacency of vertices within the corpus. So a chromosome consisted of multiple adjacency-matrix index numbers to represent the graph, and the index numbers referred to a directed edge where the node was an ROI. Index number 1 meant node 2 is connected to node 1 where node 2 is the parent node of node 1. A root node was defined as

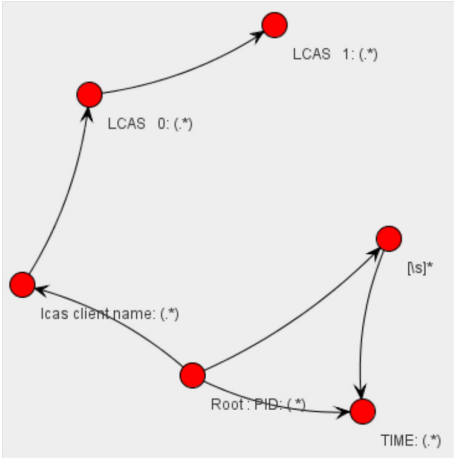

Fig. 4. Example of a directed graph, representing a corpus

a starting point for the corpus (in this case PID). In sum, first a root node was defined as a starting point for the graph. Next, the adjacency index numbers were translated into a graph of regular expressions.

Table 2. Example of an adjacency matrix. Parent node 2 has child node 3.

u / v	Node 1	Node 2	Node 3
Node 1	0	1	2
Node 2	3	4	5
Node 3	6	7	8

The genetic algorithm fitted each graph (chromosome) onto the manifest, where the root node was taken as the starting point of the graph. After a fit of the root node it tried to fit the first child node on each following line of the manifest. After a match, the next child node was fitted, and so on. This recursive process was repeated until the function reached the end of the manifest or until the root node fitted again. The latter was taken as a re-occurrence of the pattern.

A function *ScoreGraph* was used to return the longest path, or fit of nodes, between two root nodes. The *ScoreGraph* function returned the best path including recursion of the directed graph (chromosome), and was defined as

$$ScoreGraph = argmax(followedpaths)$$

When the whole manifest was covered and scored -by fitting the graph- the total score was summed. A fitness score was calculated, where n was the fit of a root node, with $k = 1$ summing the scores of *ScoreGraph*.

$$FitnessScore = \sum_{k=1}^{n} ScoreGraph^{k}$$

For optimizing to the shortest path description, the Occam's razor [12] principle was used. For every gene-space in the chromosome that was not filled, a bonus of 0.1 was added to the fitness score of the chromosome. As such, argmax tried to create the longest description possible, while Occam's razor principle tried to create the shortest possible description. Together these combined forces resulted in a complete, as well as shortest path description. This created a chromosome that represented the shortest reoccurring data pattern in the manifest, given the knowledge from the solution domain.

Precision and recall were used as evaluation measures for validating an ROI. Precision is a measure of exactness or fidelity meaning nothing but the truth:

$$Precision = \frac{|TP|+|FP|}{|TP|}$$

Recall is the probability that a (randomly selected) relevant item is retrieved in a search.

$$Recall = \frac{|TP|}{|TP|+|FN|}$$

The true positives (TP) represented the number of detected correct matches, false positives (FP) were the number of falsely detected matches, false negatives (FN) represented the number of unidentified data that were in fact real matches, and true negatives (TN) were the number of unidentified data that were not matches.

First, the genetic algorithm was tested using the manifest as input data, which contained 524 lines of entries. The amount of TPs within the manifest was known before hand, and consisted of 151 TPs.

The fitness function used a random mutation operator. A random mutation operator means a new random mutation rate was created for each evolution and applied to the current generation of chromosomes. Chromosome size and population size were manipulated. Six configurations were tested, and al configurations were tested 5 times (runs) over 300 evolutions. In this case the mutation operator was randomized every evolution:

1. Chromosome size = 11 (equal to the ultimate adjacencies), population size = 50
2. Chromosome size = 11 (equal to the ultimate adjacencies), population size = 100
3. Chromosome size = 15, population size = 50
4. Chromosome size = 15, population size = 100
5. Chromosome size = 20, population size = 50
6. Chromosome size = 20, population size = 100

3.1 Multi-agent System

The prototype was tested using several agents that had limited knowledge about a log file. The goal of the prototype was to show that agents could learn from

one another by exchanging domain knowledge and structure knowledge. In the current setting, four agents were given the same log file to analyse while holding a different set of DAs.

Each agent was given the location of the manifest and contained a JADE behaviour to analyse the manifest. The genetic algorithm was implemented in the JADE agents. The prototype uses a minimal vocabulary (JADE ontology) for communication between agents, which consisted of *corpusscore*, *score*, *da*, *name*, *expression*, *ihavescore*, and *ihaveda*.

CorpusScore was the internal fitness score of each agent. *Score* was the score received from other agents. *DA* was an ROI, *name* was the name of the DA, and *expression* was the regular expression to represent the DA. *IhaveScore* was an attribute that indicated that an agent had a corpusScore, and finally *IhaveDA* was an attribute to indicate than an agent had one or more DAs.

Figure 5 shows the interaction between two agents. The prototype was tested using four agents located on two different machines. Each agent had different DAs but all had the same root node. Each agent built a corpus every 5 minutes and exchanged DAs according to the sequence diagram in figure 5. After an agent was created it first located and opened the manifest. Next, it created a corpus using genetic computation and its solution domain. When an agent successfully created a corpus with a fitness score on the manifest, the agent would broadcasts its fitness score to the other agents. When the incoming score (received from other agents) exceeded its own, the agent would broadcast a request for DAs. The agent with the highest score would respond with a *IhaveDA* that contained its DAs. Next, the requesting agent would add the DAs to its domain knowledge.

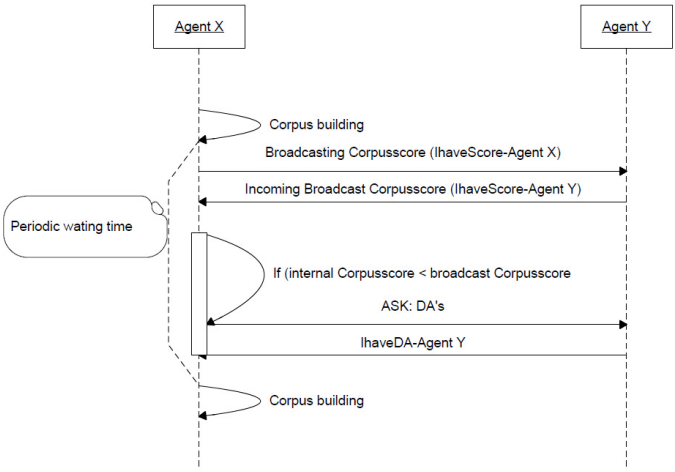

Fig. 5. Sequence diagram agent mining

4 Results

Table 3 shows the results of genetic computation with a random mutation. The max hit score was 151, and the bonus score for the shortest description was added to find the optimal corpus. In the case of chromosome size 20 the optimal score was 151.9 because 9 adjacencies should then be empty within the chromosome.

Results show that when the chromosome size was equal to the size of the optimal adjacencies (in this case 11) the optimal corpus was not found. A chromosome length of 15 required a lower number of generations to reach the optimal corpus than a chromosome size of 11 and 20. The precision score was always 1 in this test. This was because of the fact that ROIs were implemented as regular expressions. Regular expressions have a hard fit, which decreases the chance of FPs.

Two out of six configurations never reached the optimal corpus: chromosome size 11 with a population of 50 and 100. The parameter configuration of chromosome size 11 and population size 50 reached 144.1 two times. So the corpus graph score had 144 fit points, and one allele in the chromosome remained empty. This resulted in the additional bonus score of 0.1. The optimal score was 151 and all the alleles in the chromosome should be filled. Results show that the maximum fitness score was reached at an average of 90 evolutions, with an average fitness score of 140. With a population size of 100 and chromosome size of 11, the average highest fitness score was 141.3, which was reached after an average of 25 generations. It suggested that a higher population size resulted in a higher fitness score compared to a population of 50. In addition, the highest score was also reached at an earlier stage of evolution. With a population of 50, an average of 140 evolutions was needed to reach the highest score, and with a population of 100 the result was an average of 25 evolutions. A population of 100 reached its maximum score 115 evolutions sooner, compared to a population of 50. These results suggest that using a chromosome size equal to its optimal adjacencies (in this case 11, known from the manifest) did not result in the optimal corpus, and tended to reach local optimal solutions.

Using the parameters of chromosome size 15 and population 50 resulted in an average evolution number of 116 before reaching its average maximum fitness score of 150. During 4 out of 5 runs the optimal corpus was found. With a population size 100, the average fitness score was approximately 149 and the average evolution number was approximately 40, and 3 out of 5 runs reached the optimal corpus.

A chromosome size of 20 and population of 50 resulted in the optimal corpus during all runs of the experiment. In this case the average evolutions (resulting in a maximum score) was 201.6. Both cases resulted in a recall and precision of 1. When the population was set to 100 the average of evolutions that reached its maximum fitness score was approximately 99.

Results suggested that doubling the chromosome size reduced the chance of local optima solutions. Also, a higher starting population seemed to reduce the number of evolutions required to reach the optimal corpus. In this experiment chromosome size 20 and population 100 showed the best result for acquiring the optimal corpus (without respect to computation time).

Table 3. Genetic computation result with a random mutation

Chromosome size	Population size	Run	Start fitness evolution at 1	Max. fitness score	Max fitness score at evolution	Ultimate corpus found	Precision	Recall
11	50	1	79.0	137.2	37	NO	1	0.907
11	50	2	99.1	144.1	101	NO	1	0.954
11	50	3	72.0	130.3	23	NO	1	0.861
11	50	4	89.0	137.1	261	NO	1	0.907
11	50	5	99.0	144.1	25	NO	1	0.954
15	50	1	96.0	151.4	56	YES	1	1
15	50	2	85.1	151.4	76	YES	1	1
15	50	3	106.0	151.4	94	YES	1	1
15	50	4	96.0	151.4	279	YES	1	1
15	50	5	106.0	144.5	73	NO	1	0.954
11	100	1	99.0	144.1	18	NO	1	0.954
11	100	2	89.0	144.1	26	NO	1	0.954
11	100	3	89.1	144.1	21	NO	1	0.954
11	100	4	106.0	144.1	30	NO	1	0.954
11	100	5	99.0	130.3	27	NO	1	0.861
15	100	1	106.0	151.4	32	YES	1	1
15	100	2	113.0	151.4	28	YES	1	1
15	100	3	99.0	144.5	65	NO	1	0.954
15	100	4	99.0	144.5	38	NO	1	0.954
15	100	5	113.0	151.4	35	YES	1	1
20	50	1	106.0	151.9	274	YES	1	1
20	50	2	106.0	151.9	225	YES	1	1
20	50	3	103.1	151.9	197	YES	1	1
20	50	4	113.0	151.9	200	YES	1	1
20	50	5	96.0	151.9	112	YES	1	1
20	100	1	113.0	151.9	121	YES	1	1
20	100	2	116.1	151.9	95	YES	1	1
20	100	3	113.0	151.9	132	YES	1	1
20	100	4	120.1	151.9	63	YES	1	1
20	100	5	116.1	151.9	82	YES	1	1

The prototype illustrated that agents were able to learn from each other. Over time each of the four agents was able to build identical and optimal corpora, because they shared all the DAs known in the agent network.

5 Discussion

A MAS prototype was created to exchange domain knowledge between agents. This exchanged knowledge was then used by each agent to create a corpus for information retrieval from log-file manifests. The aim of the prototype was to investigate whether such an approach has the potential to support domain administrators in grids during error tracing.

The created prototype was able to successfully exchange knowledge between agents. Each agent was then able to use this knowledge to create an optimal corpus for information retrieval. The prototype allowed extraction of data from log-files even when the structure of the log-files changes over time, and new or changed domain knowledge can be introduced and shared easily among agents. The prototype therefore illustrated collaborative learning and the automatic integration of knowledge for reading manifests, and showed the potential to support information retrieval in a cross-domain volatile environment.

The prototype represented a first step towards communities of agents that will collaborate to learn log-file structures and exchange knowledge across organizational domains. Some limitations of the prototype will have to addressed in follow-up research. For instance, the current prototype was successful when analysing log files that were similar in structure and contained similar ROIs. Future work will have to address comparison of log-files in different cluster organisations, which contain different entry orders as well as ROIs that are different in nature. The latter would require semantic translation and comparison between domain knowledge of cluster organisations. For instance PID in one cluster domain may be represented as ID in another. The matching of ROIs on a semantic level was outside the scope of the current prototype.

Finally, the current study did not address scalability, deployment, and overhead of the prototype in a production cluster. While the JADE framework is known for its scalability and deployment in distributed systems [11] the genetic computation that was used could potentially increase overhead. Impact on system performance and optimization should be addressed in future work.

Acknowledgements. This study was performed within Collaborative Network Solutions (CNS), an expertise group at Logica, in collaboration with NIKHEF. The study was funded by the VL-e project (www.vl-e.org).

References

1. EGEE: EGEE Homepage, `http://public.eu-egee.org/`
2. Mulder, W., Jacobs, C.: Grid management support by means of collaborative learning agents. In: Proceedings of the 6th International Conference Industry Session on Grids Meets Autonomic Computing, pp. 43–50. ACM (2009)

3. Oliner, A., Ganapathi, A., Xu, W.: Advances and challenges in log analysis. Communications of the ACM 55, 55–61 (2012)
4. Russell, S., Norvig, P.: Artificial Intelligence: A modern approach, 3rd edn. Prentice-Hall, New Jersey (2009)
5. Cao, L., Gorodetsky, V., Mitkas, P.A.: Agent Mining: The Synergy of Agents and Data Mining. IEEE Intelligent Systems 24(3), 64–72 (2009)
6. Cao, L.: Data Mining and Multi-agent Integration (edited). Springer (2009)
7. Cao, L., Weiss, G., Yu, P.S.: A Brief Introduction to Agent Mining. Journal of Autonomous Agents and Multi-Agent Systems 25, 419–424 (2012)
8. Feldman, R., Sanger, J.: The text mining handbook: advanced approaches in analyzing unstructured data. Cambridge University Press (2007)
9. Koza, J.R., Keane, M.A., Streeter, M.J., Adams, T.P., Jones, L.W.: Invention and creativity in automated design by means of genetic programming. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 18, 245–269 (2004)
10. Conrad, E.: Detecting Spam with Genetic Regular Expressions. SANS Institute Reading Room (2007), http://www.giac.org/certified_professionals/practicals/GCIA/0.793
11. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing multi-agent systems with JADE. Wiley (2007)
12. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K.: Occam's razor. Information Processing Letters 24, 377–380 (1987)