4th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS Miami, Florida, USA, November 17-19, 2005 (pp238-241)

Computational intelligence agent-oriented modelling

Roman Neruda
Institute of Computer Science
Academy of Sciences of the Czech Republic
P.O.Box 5, 18207 Prague, Czech Republic

Abstract: In this paper a multi agent platform targeted towards the area of computational intelligence modeling is presented. We show the design of various computational agents and multi agent systems, as well as the infrastructure capabilities. The focus of the system is the interchangeability of computational components, their autonomous behavior, and emergence of new models. It is demonstrated that such a system is able to assist in building artificial intelligence models based on data in a distributed environment.

Key–Words: Multi-agent systems, Adaptive Agents, Computational Intelligence.

1 Introduction

Hybrid models, including combinations of artificial intelligence methods such as neural networks, genetic algorithms and fuzzy logic controllers, seem to be a promising and extensively studied research area [4]. We have designed a distributed multi-agent system [9] called Bang 3 that provides a support for an easy creation of hybrid AI models by means of autonomous software agents [6].

The use of distributed Multi-Agent Systems (MAS) instead of monolithic programs has become a popular topic both in research and application development. Autonomous agents are small self-contained programs that can solve simple problems in a well-defined domain [7]. In order to solve complex problems, agents have to collaborate, forming Multi-Agent Systems (MAS). A key issue in MAS research is how to generate MAS configurations that solve a given problem [5]. In most systems, an intelligent (human) user is required to set up the system configuration. Developing algorithms for automatic configuration of Multi-Agent Systems is a major challenge for AI research.

Besides serving as an experimental tool and a distributed computational environment, this system should also allow to create new agent classes consisting of several cooperating agents. The *MAS scheme* is a concept for describing the relations within such a set of agents. The basic motivation for schemes is to describe various computational methods. It should be easy to ‘connect’ a particular computational method (implemented as an agent) into hybrid methods, using schemes description. The scheme description should be strong enough to describe all the necessary relations within a set of agents that need to communicate one with another in a general manner.

In the following we describe the main features of our system, mainly implementation and logical descriptions of computational agents, mechanisms for verifying and proposing MAS schemes, and the decision support architecture for computational agents. The combination of these features is unique and makes *Bang* a valuable prototyping tool for computational intelligence modeling.

2 Computational Agents

Here we show, how two computational intelligence methods — artificial neural network of the RBF type, and a genetic algorithm — are represented by a MAS. This further serves for more complicated MAS schemes containing representing hybrid computational intelligence models.

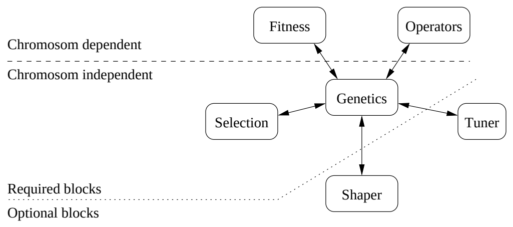

Figure 1: Genetic Algorithm as a multi-agent system.

The genetic algorithm itself, from this point of view, consists of several parts: the *Genetics* agent, which performs the basic genetic algorithm logic and glues all parts together, the *Fitness* agent, that evaluates the fitness function for each individual, the *Op-*

erators agent, that provides genetic operators, metrics operators, and creation operators, the *Selection* agent, that provides the selection of individuals. There are also two optional agent types that can further optimize overall performance: the *Shaper* agent, that provides global processing of population individuals fitness function values — such as sigma scaling — and the *Tuner* agent, that by exploiting information about the genetic algorithm performance (like best individual fitness, average fitness, first and second derivatives of these etc) tunes genetic operators probabilities (cf. Fig 1). It is supposed that there will exist more rival agents implementing a particular function (such as fitness evaluating) and it will be possible to choose among them.

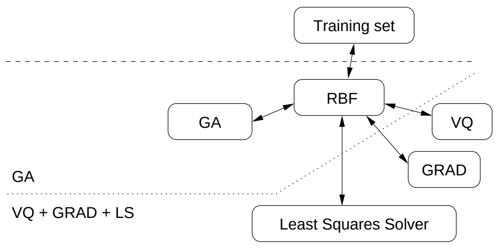

Figure 2: Radial Basis Function Network as a multi-agent system.

Similar situation is with the Radial Basis Function neural network, which is realized as a MAS containing the central *RBF* agent which cooperates with several other agents realizing training subtasks for it (cf. Fig 2). Namely, it is the *Vector quantization*, *Gradient learning*, and *Least Square Solver*, or the above mentioned *GA*. Again, it is possible that several competing agents realize a particular function for the RBF agent (we have several VQ agents based on Kolmogorov network and k-means clustering, e.g.).

3 Agents and MAS

Bang agents communicate via messages and triggers. Messages are XML documents send by an agent to another agent. A triggers are XML patterns with an associated function. When an agent receives a message matching the XML pattern of one of its triggers, the associated function is executed. In order to identify the receiver of a message, the sending agent needs the message itself and a link to the receiving agent. A conversation between two agents usually consists of a number of messages. For example, when a neural network agent requests training data from a data source agent, it may send the following messages: Open the data source located at XYZ; Randomize the order of the data items; Set the cursor to the first item; Send next item.

Multi-Agent Systems are assembles of agents (for now, only static aspects of agents are modeled). Therefore, a Multi-Agent System can be described by three elements: The set of agents in the MAS, the connections between these agents, and the characteristics of the MAS. The characteristics (constraints) of the MAS are the starting point of logical reasoning: In *MAS checking* the logical reasoner deduces if the MAS fulfills the constraints. In *MAS generation*, it creates a MAS that fulfills the constraints, starting with a partial MAS.

Figure 3 shows an example of the most simple computational MAS in *Bang* which consists only of the computational agent, data and a task manager (which can be a user interacting via GUI, or more complicated agent performing series of experiments over a cluster of workstations). A more typical computational MAS configuration is shown on figure 3, where two more complicated computational agents — an RBF network and an Evolutionary algorithm cooperating with each other — together with several simpler agents to solve a given task.

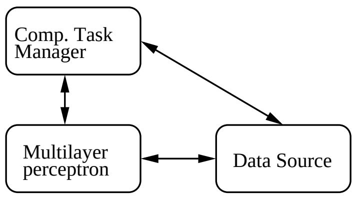

Figure 3: Example of a small computational MAS consisting of a Task Manager agent, Data Source agent, and a computational agent (Multilayer Perceptron).

4 Autonomous Behavior Support

In order to act autonomously, an agent should be able to cope with three different kind of problems [8]: cooperation of agents, a computation processing support, and an optimization of the partner choice.

Cooperation of agents: An intelligent agent should be able to answer the questions about its willingness to participate with particular agent or on a particular task. The following subproblems follow: (1) deciding whether two agents are able to cooperate, (2) evaluating the agents (according to reliability, speed, availability, etc.), (3) reasoning about its own state of affairs (state of an agent, load, etc.), (4) reasoning about tasks (identification of a task, distinguishing task types, etc.).

Computations processing: The agent should be able to recognize what it can solve and whether it is

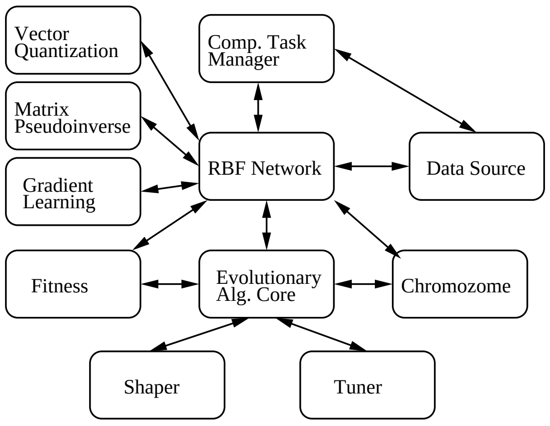

Figure 4: Example of a more complicated computational MAS consisting of a Task Manager agent, Data Source agent, and a suite of cooperating computational agents (an RBF network agent and Evolutionary agent with necessary additional agents).

good at it, to decide whether it should persist in the started task, and whether it should wait for the result of task assigned to another agent. This implies the following new subproblems: (1) learning (remembering) tasks the agent has computed in the past (we use the principles of case-based learning and reasoning — see [2], [1] — to remember task cases), (2) monitoring and evaluation of task parameters (duration, progress, count, etc.), (3) evaluating tasks according to different criteria (duration, error, etc.).

Optimization of the partner choice: An intelligent agent should be able to distinguish good partners from unsuitable ones. The resulting subproblems follow: (1) recognizing a suitable (admissible) partner for a particular task, (2) increasing the quality of an evaluation with growing experience.

So, the architecture must support *reasoning*, *descriptions* of agents and tasks (we use ontologies in descriptions logics - see, e.g., [3]), *monitoring* and *evaluation* of various parameters, and *learning*.

The architecture is organized into layers. Its logic is similar to the vertically-layered architecture with one-pass control (see [10, p. 36]). The lowest layer takes perceptual inputs from the environment, while the topmost layer is responsible for the execution of actions.

The architecture consists of four layers (see Figure 5): the *monitors* layer, the *evaluators modeling* layer, the layer for *decision support*, and the *behavior generation* layer. All layers are influenced by global *preferences*.

Global preferences allow us to model different flavors of an agent's behavior, namely, we can set an agent's pro-activity regime, its cooperation regime

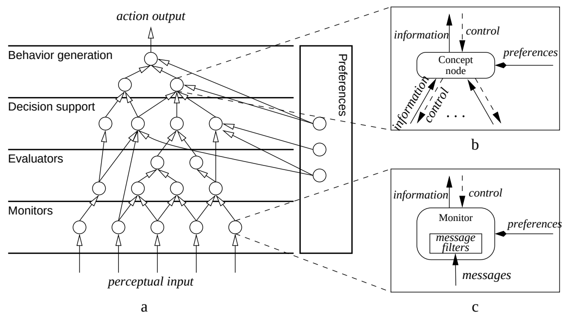

Figure 5: Architecture — network of concepts (a); Concept node (b); Monitor (c)

and its approach to reconsideration. *The monitors layer* interfaces directly with the environment. It works in a purely reactive way. It consists of rules of the form $condition \longrightarrow action$. *Evaluators modeling layer* is used to model more aggregate concepts on top of already defined concepts (either monitors or other evaluators). *Decision support layer* enables an agent to solve concrete problems. *Behavior generation layer* generates appropriate actions that the agent should perform, and thus controls the agent's behavior. The mechanisms for action generation and selection are provided by the BDI model (see [10, pages 55–61]).

5 Experiments

We have adapted two existing computational agents embedding the multi-layer perceptron (MLP) and the radial basis function (RBF) neural network. These agents represent two different computational methods for the solution of similar categories of tasks.

Overheads of the architecture are summarized in Table 1. The creation of the agent takes 2-3 times longer since all the structures must be initialized. The communication overhead is around 30% when dealing with message delivering. However, in real-life scenario of task solving, the overhead is only about 10%.

	Without the arch.	With the arch.
Agent creation time	3604 μs	9890 μs
Message delivery time	2056 μs	2672 μs
Total computation time	8994681 μs	9820032 μs

Table 1: Comparison of the agent with and without the autonomous support architecture

Table 2 summarizes the measured results of *opti-*

	Error	Duration
Random choice	11.70	208710ms
Best speed	1.35	123259ms
Best Accuracy	1.08	274482ms
Best services	1.17	102247ms

Table 2: Optimization of the partner choice. Comparison of choices made by different criteria.

Repeated tasks	Standard	Optimized
0 %	135777097	121712748
20%	94151838	90964553
40%	50704363	91406591
60%	47682940	90804052

Table 3: Optimization by reusing the results of previously-computed tasks (duration in milliseconds).

mization of the partner choice. We simulated a usual scenario when an agent needs to assign some tasks to one of admissible partners. This agent uses a collection of different tasks and assigns them to the computational agents successively. The total duration of the computation and the average error of computed tasks were measured. A significant improvement of the efficiency can be seen.

Experiments with *optimization by reusing results* are summarized in Table 3. We have constructed several collections of tasks with different ratios of repeated tasks (quite a usual situation when, e.g., evaluating the population in genetic algorithms). We compared the total computation-times of the whole collection with and without the optimization enabled. We can see that the optimization is advantageous when the ratio of repeated tasks is higher than 20%. When more than 40% are repeated the results are significant.

6 Conclusions and Future Work

We have demonstrated that *Bang* is able to help both scientists and end-users with data analysis tasks. The niche for this software has been prototype building and testing various hybrid models so far. However, it is possible to employ it for large scale distributed computations running on a cluster of workstations. The nature of evolution of MAS schemes has brought several issues that are currently being solved. The resulting hybrid search for MAS solution to a particular problem represented by data should be not only automatic, but also feasible in terms of computational time and resources consumption.

Acknowledgement

This work was supported by the Ministry of Education of the Czech Republic under the project Center of Applied Cybernetics No. 1M684077004 (1M0567).

References:

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning : Foundational issues, methodological variations, and system approaches. *AICom — Artificial Intelligence Communications*, 7(1):39–59, 1994.

[2] David W. Aha and Dietrich Wettschereck. Case-based learning: Beyond classification of feature vectors. 1997.

[3] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[4] P. Bonissone. Soft computing: the convergence of emerging reasoning technologies. *Soft Computing*, 1:6–18, 1997.

[5] J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[6] S. Franklin and A. Graesser. "Is it an agent, or just a program?": A taxonomy for autonomous agents. In *Intelligent Agents III*, pages 21–35. Springer-Verlag, 1997.

[7] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(2):205–244, 1995.

[8] Roman Vaculin and Roman Neruda. Concept nodes architecture within the Bang3 system. Technical report, Institute of Computer Science, Academy of Science of the Czech Republic, 2004.

[9] G. Weiss, editor. *Multiagent Systems*. The MIT Press, 1999.

[10] Gerhard Weiss, editor. *Multiagents Systems*. The MIT Press, 1999.