Hybrid AI Models using Bang

Krušina, Pavel
2002

Institute of Computer Science
Academy of Sciences of the Czech Republic

Hybrid AI models using Bang

Pavel Krušina and Petra Kudová and Zuzana Petrová
and Roman Neruda

Technical report No. 879

October 2002

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: (+420) 26605 3750, fax: (+4202) 865 85 789,
e-mail:bang@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Hybrid AI models using Bang

Pavel Krušina and Petra Kudová and Zuzana Petrová
and Roman Neruda [1]

Technical report No. 879

October 2002

Abstract:

We describe a system which represents hybrid computational models as communities of cooperating autonomous software agents. It supports easy creation of combinations of modern artificial intelligence methods, namely neural networks, genetic algorithms and fuzzy logic controllers, and their distributed deployment over a cluster of workstations.

Keywords:
Multiagent systems, hybrid models

[1] This work has been partially supported by GACR under grant 201/00/1489.

1 Introduction

Hybrid models, including combinations of artificial intelligence methods such as neural networks, genetic algorithms or fuzzy logic controllers, seems to be a promising and currently studied research area [6].

We have designed a distributed multiagent system called Bang, that provides a support for an easy creation of hybrid AI models by means of software agents. As an 'agent' we understand an entity, which is autonomous, reacts to its environment in pursue of its own agenda [8]. The agent can be adaptive, or intelligent in a sense that it is able to gather information it needs in some sophisticated way. Moreover, our agents are mobile and persistent.

In the next section we introduce the basic ideas of Bang, section 3 outlines some agents designed in our system and already being used, than the conclusion follows, where the current state and future work are described.

2 Bang

Bang [1] is a library for various artificial intelligence methods and environment for hybrid AI models development. It consists of the population of agents living in an environment. The environment provides the necessary support for Bang's run such as creation of agents, giving them information necessary to survive and be able to communicate (e.g. where are other agents), distribution processes to their computational nodes (parallelism, load balancing). It also delivers messages and transfers data.

2.1 Implementation

The current version of implementation is called Bang3. It is a group of environments for hosting common agents. By narrowing the interfaces between an agent and an environment, respectively utility functions and an agent, we hope to allow running one agent in several environments and to have utility functions independent on the environment implementation.

The minimal, single-threaded, single-process, non-distributed, synchronous-only environment is called Divine Offering. It is the most simple implementation of Bang environment able to run Bang3 agents. It is easy to use as a computation library for other programs and for batch mode data processing when the suitable model is found. It is also easily portable to a large variety of computer platforms.

The ultimate Bang3 environment, called Stronghold, is in progress. It is distributed, multi-threaded and multi-process. It is able to run each agent in a separate process when this is desired for easy debugging or to run more agents in one process when the highest performance is the concern.

An agent in the Bang3 is a living object. You can think of it as a object with its own thread, memories and desires. An agent communicates with other agents using messages in a XML-derived language. We call it "send a message", because you prepare a message in some language and then pass it to the other agent. In fact the implementation decides, independently of your code, whether to send messages somewhere or to simply call the other agent's method.

In fact an agent is represented by a C++ class derived from the parent Agent class. When a message arrives an appropriate member function (trigger) is called.

To simplify message handling we provide an extended syntax (see figure 2.1), that allows an easy mixing C++ code with a simple notation of XML-derived messages (trees). Our Perl preprocessor scans your source code and generates pure C++.

2.2 Communication language

There is a need of an inter-agent communication language. The language should cover constructs for communication between agents as well as system messages and be able to describe arbitrarily structured data.

Nowadays a lot of agent communication languages appear. We were inspired by KQML [7] (de facto standard for an agent communication) and ACL [3]. Let's also mention XSIL [10] and PMML [4], languages using XML for sending structured data through net.

```
  /*  launcher code */
DefReturn OK() ( ok )
DefReturn UGH() ( ugh )

Trigger( request launch $CString::AgentName )
{
    if(!CreateAgent( AgentName )) OK;
    else                  UGH;
}

  /* caller example */
 CMsg mesg = BuildMsg( request launch $AgentName );
 CMsg ret = Sync(pLauncher, mesg);

 { Switch ( ret ) {
    Case ( ok ) : {
      puts("Ok from Launcher");
  }
    Case ( ugh ) : {
      puts("Ugh from Launcher");
    }
  }} //end switch
```

Figure 2.1: Extended syntax example.

Our language is based on XML. A message is represented by a tree of tagged nodes. The type of the message is given by the root of the tree. There are several types of messages, namely `<request>`,`<query>`,`<inform>`,`<ok>` (positive answer),`<ugh>` (negative answer, error indication).

Some messages are comprehensible to all agents (answering to them is implemented by the parent Agent class), mostly system messages like *kill* (yourself),*move* (to another computer), and some useful messages like *ping* (Are you alive?), other only to a special group of agents (e.g. neural nets).

3 Agents in Bang

Now we introduce some agents, that have already been designed by means of Bang. First of all, there are some agents implementing fundamental components of the system. Second, there are also some AI applications, namely an implementation of a general genetic algorithm and algorithms for learning of RBF networks [9, 2].

3.1 System agents

While environment does services necessary for the system to work, useful and less critical services are provided by system agents. So there is a possibility of replacing a system agent with another one and thus make system more configurable. We will mention two of the most important system agents: Yellow and White Pages.

Yellow Pages are the manager of all running agents. Each agent immediately after creation should inform Yellow Pages about its name, identification number and capabilities. Yellow Pages are then able to find all agent with desired capabilities.

White Pages are manager of all agent states. This allow our agent to be persistent. It stores a state of the agent with some information about its capabilities and types. (Type is e.g. MLP, capabilities is what it is learned for, e.g. recognition of text).

Another system agents are in development phase, e.g. load balancing agent, monitoring the load and system configuration of the computers in the clusters and advising what computers are free and suitable for desired computation.

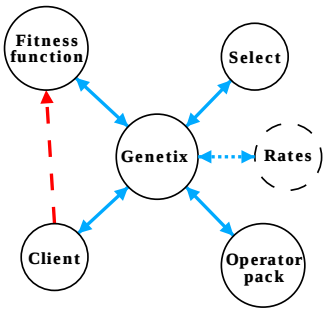

Figure 3.1: Implementation of genetic algorithm.

3.2 Genetic algorithm

The central point of the GA implementation in our system is the Genetix agent (see figure 3.2). It is the abstraction of the GA-based solvers, concentrating the common things and leaving all the problem specific code to the others. Genetix cooperates with several other agents: the fitness function calculating agent for evaluating the solution of the given problem; the genetic operators agent for modifying the solution candidates; the selector agent for easy change of the selection mechanism; and optionally with the agent tuning the GA operators rates.

Let us try to implement a GA-based solving method of some our problem into Bang. First we need to choose data representation of the problem solution candidates (e.g. vectors of bits or matrices of floats etc). Next we need an operator package agent operating on the chosen data representation. Then, the selector agent is needed, in most cases the roulette wheel agent is suitable. We also can connect to the Genetix the agent tuning the operator rates during the learning process, but in most cases setting them manually in the user-dialog window is sufficient. Now we have the generic GA working on the chosen data representation. The final step is to (program and) connect the fitness function computing agent, which by giving marks to the solution candidates directs the Genetix towards our desires.

In Bang we have a modular and open implementation of GA allowing for easy extending and code reusing and minimizing and simplifying the task of creating the new problem GA-based solver.

3.3 RBF networks

There is a variety of learning methods for RBF networks, and in addition there are several variants of RBF units including a choice of a radial function. We have decided to implement the RBF network as a combination of the network agent and a family of learning agents. The network agent represents the network itself, we will call it the RBF Agent. It is very poor at learning knowing only some simple or straightforward methods. But it is intelligent in the way that it lets other agents do the real work. On the other hand, the learning agents do not make any decisions about what algorithm to use for learning, but they are very good at solving the learning problem using their own method. They can be connected to the RBF Agent and do its partial learning.

In figure 3.3 we see our agents cooperating on learning of an RBF network. Trainset is a special agent implementing the interface to a training set. All agents can be connected to the GUI and configured through it.

In the tree-step learning method, the problems of vector quantization and least squares turn up. Therefore we have agents for solving these particular problems. These agents are independent of a RBF network. It means that they can be used by whatever agents that need to solve an instance of the vector quantization problem or the least squares problem.

Another type of agents is represented by the gradient method for the 2nd step and the gradient learning. The former is designed for setting parameters of RBF units (widths, norm matrices). It uses the gradient method for a minimization of the error function. The latter solves the learning problem by modification of the back propagation algorithm. They can be both used only for RBF networks, nowhere else. But they can be configured for several purposes, they can learn all parameters, or only some of them.

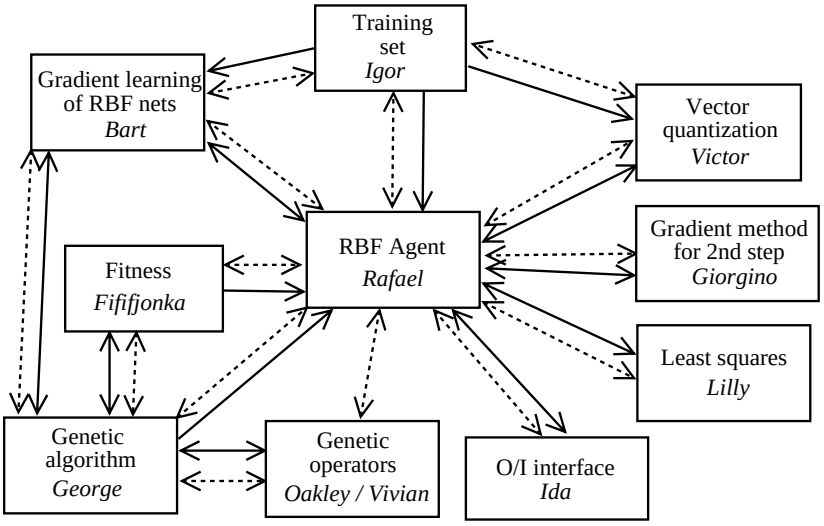

Figure 3.2: Agents cooperating on an RBF learning. Dotted arrows represent a flow of messages, solid arrows binary data interchange.

The Genetix (described earlier) can be also useful here, using appropriate fitness agent and genetix operators. It can be used for learning all RBF network parameters or only for some of them determining the rest by another method.

During experiments a combination of the three-step method and the gradient learning turned up to be very powerful. Since the three-step method is rather quick, it can be used for initialization of parameters for much slower but much more precise gradient learning.

The system described is open in the sense of adding new agents and learning methods. Any existing agent can be replaced by a different agent with the same interface. So the learning system is hoped to be further developed and improved.

4 Conclusion

We described the distributed multiagent system called Bang, as an open system useful for an easy creation of autonomous software agents with respect to a design of hybrid AI models.

The system is still being developed. We have already some agents realizing particular AI methods, let us mention genetic algorithm, group of RBF learning methods and MLP with Back Propagation [5].

In our future work we will focus on mirroring agents, parallel execution, automatic scheme generating and evolving. Also the concept of an agent working as the other agent's brain by means of delegating the decisions seems to be promising. Another thing is the design of load balancing agent able to adapt to changing load of host computers and to changing communication/computing ratio.

Bibliography

[1] *Bang Homepage*. `http://bang2.sourceforge.net/`.

[2] *Learning methods of RBF networks (www)*. `http://atrey.karlin.mff.cuni.cz/~petra/rbf/`.

[3] Agent communication language. Technical report, Foundation for Intelligent Physical Agents, 1998.

[4] PMML v1.1 predictive model markup language specification. Technical report, Data Mining Group, 2000.

[5] Jakub Adámek. Neural networks controlling prosody of czech language. Master's thesis, The faculty of mathematics and physics, Charles University in Prague, 2002.

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

[7] Tim Finnin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. *Software Agents*, 1997.

[8] 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.

[9] Petra Kudová. Learning methods of RBF networks (in czech). Master's thesis, The faculty of mathematics and physics, Charles University in Prague, 2001.

[10] Roy Williams. Java/XML for scientific data. Technical report, 2000.