Multiagent Environment for Hybrid AI Models

ROMAN NERUDA, PAVEL KRUŠINA, ZUZANA PETROVÁ*
Institute of Computer Science
Academy of Sciences of the Czech Republic
P.O. Box 5, 18207 Prague, Czech Republic

Abstract: To utilize the the concept of combining various modern artificial intelligence methods — namely neural networks, genetic algorithms and fuzzy logic controllers — we develop a unified software platform called Bang2. The system serves as a library for many artificial intelligence methods and allows for easy creation of such combinations, and possibly their semi-automated generation, or even evolution. It also ensures their deployment through the computer network and parallel processing. Inspired by software agents paradigm we have designed Bang2 as a community of cooperating autonomous software agents.

Key-Words: Hybrid methods, Multiagent systems, Adaptive agents.

1 Introduction

Since the practical use of artificial intelligence methods, such as neural networks, genetic algorithms as well as their simple combinations seem to be widely explored ([1]), we have turned our effort to more complex combinations, which are out of focus of most researchers, probably also because of the lack of a unified software platform that would allow for experiments with hybrid models.

Design of Bang2 pursues two goals. At first to serve as a library for many artificial intelligence methods and thus help developers to design their own applications. Moreover the unified interface allows to switch easily e.g. between several learning methods, and to choose the best combination for application design. Parallel processing is the expected and useful advance here as well as a rapid and easy design. Second goal of Bang2 design involves creation of more complex models, semi-automated models generation and even evolution of models.

For distributed and relatively complex system like Bang2 it is favorable to make it modular and to prefer the local decision making against global intelligence, and therefore to take advantage of agent technology. Employing software agents also simplifies the implementation of new AI components (unified interface) and addition of them without recompiling and restarting.

2 Overview of Bang2

Bang2 consists of a population of sundry agents living in the Environment. The Environment provides the necessary support for Bang2'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.

Agents are the basic building blocks of Bang2. Each agent provides and requires services (e.g. statistic

*This work has been partially supported by the Grant Agency of the Czech Republic under grants no. 201/00/1489 and 201/99/P057.

agent provides statistic preprocessing of data and requires data to process). Agents communicate via special communication language encoded in XML. There are several special agents necessary for Bang2's run (like the Yellow Pages agent maintains information about all living agents and about the services they provide). Other (not special) agents do the real work (read data from files, represent neural net, learn neural net, provide numeric calculation for other agents etc.)

Before any further insight to Bang2 the term agent should be approached. Exact definition of agent doesn't exist, every group using agents provides its own definition. For introduction to software agents see [3]. Generally software agent is a computer program, which is autonomous, reacts to its environment (e.g. to user's commands or messages from other agents) and when nothing interesting happens it doesn't wait for the next event as regular program, but does its own work. It usually follows its own goal. It is adaptive and intelligent in sense that it is able to obtain information it needs by asking somebody (other agent, a human, a server). Moreover it is usually mobile, persistent, and sometimes tries to simulate human character.

3 Architecture

The Bang2 system consist of two fundamental parts — the environment and the agents. The environment serves as a living space for all the agents, giving them resources they need and serves as a communication layer. These are the main aspects we want to keep on mind when designing and programming the environment:

Abstraction — hiding of raw hardware and OS to our agents and providing most of services and resources in friendly and comfortable manner.

Transparency — hiding as much implementation details as possible and suitable while still allowing agents to explicitly request such informations. The first task the transparency comes to our mind is communication — the goal is to make the communication being simple for the agent programmer and exactly the same for local and remote case while still exploiting all the advantages of the local one.

Scalability — hope to design and write program with no built-in limits of amount of usable resources. We want our program to be run on computers of very different performance: from small laptops for agents programmers to huge clusters for real number crushing.

Adaptability — ability to run agent schemes developed on small systems on huge ones and vice versa. Preferably with only small need of manual interference.

Helper functions — being friendly to agent programmer. Insert a lot of functionality to agent base class and provide a code generators.

3.1 Communication layer

What we call communication layer is mainly the environment and a small code in agent base class. Purpose of it is to allow communication between agents. What we expect from it:

Simplicity — we want the communication to be simple from the agent programmer's point of view, something like a single function call.

Location transparency — there should be no difference for the agent programmer between communication to local and remote agent.

Synchronicity – we want to provide an easy way how to select synchronous, asynchronous or deferred synchronous mode of operation for any single communication act.

Efficiency — we want to be efficient both in passing XML strings and binary data.

As the best abstraction for the agent programmer we have chosen the model of object method invocation. Among its advantages let us mention the facts that programmers are more familiar with concept of function calling then message sending and that the model of object method invocation simplifies the trivial but much

Medium	XML strings	CData*	function parameters
Call	Sync	BinSync	UFastNX
Generality	High	Run-time	Hardwired
Speed	Normal	Fast	The fastest

Table 1: Communication functions properties: Sync is a blocking call of the given agent returning its answer, Async is non-blocking call discarding answer and Dsync is non-blocking call storing answer at negotiated place. BinSync and BinDsync are same as Sync and Dsync but the exchange binary data instead of XML strings. UFastNX is a common name for set of functions with number of different parameters of basic types usually used for proprietary interfaces.

common cases while keeping the way to the model of message passing open and easy. Sync is a blocking call of the given agent returning its answer, Async is non-blocking call discarding answer and Dsync is non-blocking call storing answer at negotiated place. BinSync and BinDsync are same as Sync and Dsync but the exchange binary data instead of XML strings. UFastNX is a common name for set of functions with number of different parameters of basic types usually used for proprietary interfaces.

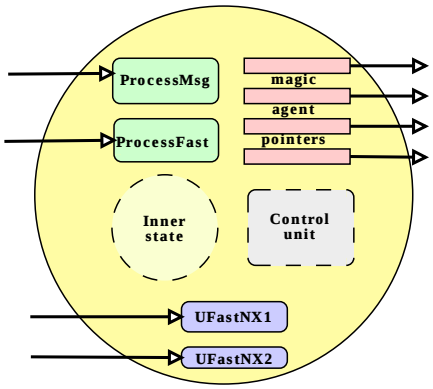

Figure 1: Agent inwards

```
TRIGGER1( request_setfriend, id )
{
  SETAGENT( friend, id );
  OK;
}
TRIGGER0( request_pingfriend )
{
  return Sync( friend,
     "<request><ping/></request>" );
}
```

Figure 2: Triggers code

3.2 Agents

All agents in Bang2 are regular C++ classes derived from base class Agent which provide common services and connection to environment (Fig. 1). Each agent behavior is mainly determined by its ProcessMsg functions which serves as main message handler. The ProcessMsg function parses the given message, runs user defined triggers via RunTriggers function and, if none is found, the DefaultBehavior function is called. The DefaultBehavior function provides standard processing of common messages. Agent programmer can either override ProcessMsg function on his own or (preferably) write trigger functions for messages he want to process (Fig. 2). Triggers are functions with specified XML tags and attributes. RunTriggers function calls a matching trigger function for a received XML message and fills up the variables corresponding to specified XML attributes with the values (see 4).

Magic agent pointer is in fact an association of a regular pointer to Agent object with a string containing its stringified handle registered to the Agent class, so DefaultBehavior function can automatically adjust the pointer and the handle according to information emitted by Black Pages.

Finally inner state is a general name for values of relevant member variables determining the mode of agent operation and its learned knowledge. The control unit is its counterpart — program code manipulation with the inner state and performing agent behavior, it can be placed in all ProcessMsg/ProcessFast functions or triggers.

4 Communication language

Consider a simple example that iterates various stages in a design of an agent communication. Let's have a neural net. It looks around for a learning agent, negotiates with other agents if they are able to learn it, are free (do not learn any other agent already) and negotiates format of data to transfer. Then these two agents connect together and exchange data (neural nets weights and error). We need to:

- **specify message headers**. Should be human readable.
- **language for agent messages** (negotiation, control sequences). Should be human readable, declarative.
- **language for data transfer**. Should be able to transfer complex data structures through simple byte stream.

There are several languages for these purposes. ACL ([4]) and KQML ([2] — widely used, de facto standard) define (among others) format of message headers and communication protocols. They are lisp-based.

KIF (KQML group — [5]), ACL-Lisp (ACL group — [4]) are languages for data transfer. They both came out of predicate logic and both are lisp-based, enriched with keywords for predicates, cycles etc. XSIL [8] and PMML [7] are XML-based languages designed for transfer of complex data structures through the simple byte stream.

Messages in Bang2 system are syntactically XML strings. Headers are not necessary, because the inner representation of messages (method invocation), so the sender and receiver are known. First XML tag defines the type of the message (similar to message type defined in an ACL header). Available message types are: *request, inform, query, ok* (reply, no error), *ugh* (reply, an error occurs).

The rest of the message (everything between outermost tags) is the content. It contains commands (type request), information provisions, etc. Some of them are understandable to all agents, others are specific to one agent or a group of agents.

There are two ways how to transfer data:

```
<broadcast><halt/></broadcast>
<inform>
<created myid="!000000000001"
  name="Lucy"
  type="Neural Net.MLP"/>
</inform>

<ok>Agent Lucy, id=!000000000001,
type=Neural Net.MLP created</ok>

<request><ping/></request>
```

Figure 3: Example of Bang2 language for agent negotiation

```
<query><vector row="45"/></query>
<query><vector/></query>
<ok><data separator=",">
Here are binary data
</data></ok>
<query><bin><query>
<vector/>
</query></bin></query>
<ok session="5" funcnum="1"/>
```

Figure 4: Example of Bang2 language for data transfer

- **As a XML string** — human readable, but lack performance (the lack of performance is not fatal in agents' negotiation stage (as above), but is a great disadvantage when they're transferring data).
- **As a binary** — much quicker, but receiver have to be able to decode it.

Generally in Bang2 the XML way of data transfer is implicit and the binary way is possible after the agents make an agreement about the format of transferred data.

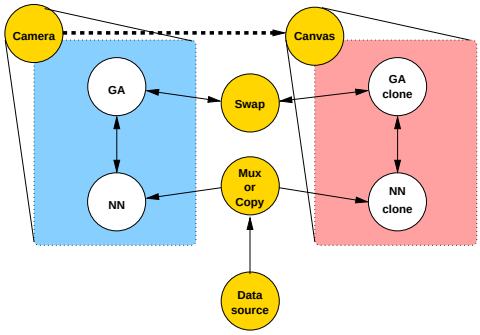

Figure 5: Task parallelization

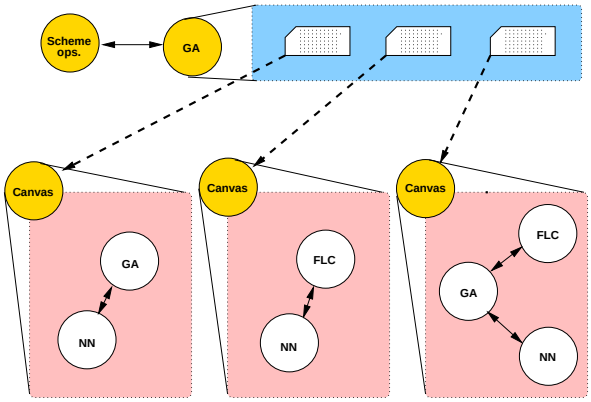

Figure 6: Scheme evolving

5 Conclusion

For now, the design and implementation of the environment is complete. So, we have started to create a set of agents of different purpose and behavior to be able to start designing and experimenting with adding more sophisticated agent oriented features to the system. There are going to be GA and RBF agents in near future.

For experimenting with agent schemes, we need agents of various types. We want to try mirrors, parallel execution, automatic scheme generating and evolving. Also concept of an agent as the other agent's brain by means of decisions delegating 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. To make interaction with human more comfortable we want to create a user-friendly graphical user interface. Preferably in way allowing easy swap to non-graphical representation. And finally we think about some form of inter Bang2-sites communication.

In the following we discuss some of these directions in more details.

5.1 Task parallelization

There are two ways of parallelization: by adding ability to parallelize its work to a computation agent or by creating generic parallelization agent able to manage non-parallel agent schemes. Both have their good and weak sides, but here is no reason not to implement both and let the user or agent programmer to choose. Consider an example of a genetic algorithm. It can explicitly parallelize by cloning fitness function agent and letting the population being fitnessed simultaneously. Or on the other hand, the genetic algorithm can use only one fitness function agent, but be cloned together with it and share the best genoms with its siblings via a special purpose genetic operator. We can see this in figure 5, where agents of Camera and Canvas are used to automatize the subscheme-cloning. Camera looks over the scheme we want to replicate and produces its description. Canvas receives such description and creates the scheme from new agents. You can imagine cases where each of the above approaches is better then the other, so it make sense to defer this decision till the real task is considered.

5.2 Agents scheme evolving

When thinking about implementing the task parallelization, we found very useful to have a way of encoding scheme descriptions in way understandable by regular agents. Namely we think about some kind o XML description. This leads to idea of agents not only creating and reading it, but also manipulating with it. All we need to be able to evolve agent schemes by generic genetic algorithm is to create a suitable genetic operator package. You may ask, what will be the fitness function for such genoms. The answer is simple: the part of generic task parallelization infrastructure (namely the Canvas, see fig. 6). For genetic evolving of schemes we

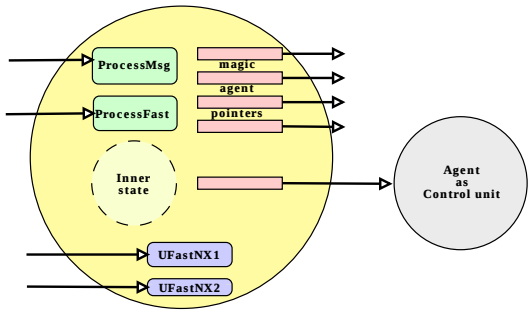

Figure 7: Agents as brains

can use the Canvas for testing newly modified schemes. In fact the only thing we want to add to be able from task parallelization advance to scheme evolving is the actual scheme genetic operator package. I find this a nice proof of reusability and good design of Bang2.

5.3 Agent as a brain of other agent

As it is now, the agent has some autonomous - or intelligent - behavior encoded in standard responses for certain situations and messages. A higher degree of intelligence can be achieved by hard-coding some consciousness mechanisms into agent. One can think of creating a planning agents, Brooks subsumption architecture agents, layered agents, or Franklin "conscious" agents. We plan to create a universal mechanism via which a standard agent can delegate some or all of its control to a specialized agent that serves as its external brain. This brain can independently seek for supplementary information, create its own internal models, etc, and finally advise the original agent what to do.

References

[1] P. Bonnisone, "Soft computing: the convergence of emerging reasoning technologies", *Soft Computing*, vol.I, pp. 6–18, 1997.

[2] Tim Finnin, Yannis Labrou, James Mayfield, "KQML as an agent communication language", *Software Agents*, MIT Press, Cambridge, 1997, http://www.cs.umbc.edu/agents/-introduction/kqmlacl.ps.

[3] Stan Franklin, Art Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", *Intelligent Agents III*, pp. 21–35, 1997.

[4] Foundation for Intelligent Physical Agents, "Agent Communication Language", *FIPA 97 Specification*, 1997, http://www.fipa.org.

[5] Michael Genesereth, Richard Fikes et. al., "Knowledge interchange format, version 3.0 reference manual", Technical Report, Computer Science Department, Stanford University, 1992. http://www.cs.umbc.edu/kse/kif/.

[6] Roman Neruda, Pavel Krušina, "Creating Hybrid AI Models with Bang", *Signal Processing, Communications and Computer Science*, pp. 228–233, 2000.

[7] "PMML v1.1 Predictive Model Markup Language Specification", Technical Report, Data Mining Group, 2000, http://www.dmg.org/html/pmml_v1_1.html.

[8] Roy Williams, "XSIL: JAVA/XML for Scientific Data", Technical Report, California Institute of Technology, 2000.