Schemes of Multiagent Systems

Rydvan, Pavel
2005

Dostupný z http://www.nusl.cz/ntk/nusl-39698

Institute of Computer Science
Academy of Sciences of the Czech Republic

Schemes of Multiagent Systems

Pavel Rydvan

Technical report No. 943

December 2005

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 051 111, fax: +420 286 585 789,
e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Schemes of Multiagent Systems [1]

Pavel Rydvan[2]

Technical report No. 943

December 2005

Abstract:

In the artificial intelligence, the hybrid models tend to provide good results for many problems. Hybrid model is a model that combines several approaches (neural networks, genetic algorithms, fuzzy logic controllers etc.). However, such a model often has many variants and parameters to be set. The success of the model depends on these options.

A scheme of the multiagent systems is an instrument for formal description and construction of the multiagent systems. Schemes are flexible enough to describe simple multiagent system as well as the hybrid model implemented by the multiagent system. This brings the potential of an automatic generation of the scheme, and therefore the potential for using the evolutionary algorithms for the scheme creation.

Keywords:
Software agents, multiagent systems, schemes, genetic algorithms

[1]This work was partially supported by GA ČR Grant 201-05-H014.

[2]Institute of Computer Science, Academy of Sciences of CR, Pod vodárenskou věží 2, P.0. Box 5, 182 07 Prague 8, Czech Republic
Charles University, Faculty of Mathematics and Physics, Ke Karlovu 3, 121 16 Prague 2, Czech Republic, pavel.rydvan@st.cuni.cz

1 Introduction

In the artificial intelligence, the hybrid models tend to provide good results for many problems [6]. Hybrid model is a model that combines several approaches (neural networks, genetic algorithms, fuzzy logic controllers etc.) [2]. However, such a model often has many variants and parameters to be set. The success of the model depends on these options.

A scheme of the multiagent systems is an instrument for formal description and construction of the multiagent systems. Schemes are flexible enough to describe simple multiagent system as well as the hybrid model implemented by the multiagent system [3]. This brings the potential of an automatic generation of the scheme, and therefore the potential for using the evolutionary algorithms for the scheme creation [5, 4].

2 Schemes

The main reason for introducing *schemes* is to provide the means for explicit description of the communication interface of both a single agent and a group of them. It is very handy to have the same instrument for describing both single agent and a group of agents, so both can be handled in the same way.

Scheme definition prescribes the two fundamental properties: *gates* and *interfaces*. While gates are the tools for sending the messages, the interfaces are the tools for obtaining messages and for distinguishing among more possible sources of them. Gates and interfaces thus are the means of communication. The connection can be established between a gate (of one agent) and the interface (of second agent).

The agent can send any message through its gate. The message is transported to the second agent's interface. The second agent can (but is not forced to) send a reply to the message. The communication is therefore "semi-bidirectional". Each message exchange is iniciated by the agent which holds the gate, but each message can be replied.

An important property of both gates and interfaces is that the target of each of them can be easily changed from outside. It is supposed, that any agent (or group of agents acting as a scheme) understands the particular messages for manipulating with gates/interfaces. With this proprty it is possible to use the schemes as a building blocks for creating *multiagent systems* (MAS).

MAS is the entity that fulfils the constrains given by the scheme definition. The simplest case of MAS is a single agent with given set of gates and interfaces. Another case of MAS is a group of agents, each of them playing its role and acting together as a single scheme. Finally, because the MAS can behave as a single agent (one can send messages to the scheme as if it was regular agent) and, similarly, a scheme can be also the originator of the messages, the MAS can be also a building block of other (bigger) MAS. On the other hand, none of the comprising agents (or MAS) looses its former ability to act on its own.

2.1 MAS

The definition of MAS is given by two its fundamental properties: the set of *building blocks* and the way the building blocks are *interconnected*.

Each building block indicates the way how to create one scheme, which takes part within the MAS. Both agent and MAS can play the role of the building block. It is therefore possible to create recursive structures where MAS is made out of other MASes and/or agents.

2.2 Building Blocks

There are two basic possibilities for describing the way how to obtain each building block in the MAS specification: the agent/MAS can be newly created when creating the MAS or a "living" agent can be connected to the MAS that is being created.

It is also possible to handle the building blocks in special ways. For example, it is possible to specify a ring of them. In this case, several "clones" of certain buiding block are created and interconnected

very easily to form the token ring, which could be very handful for parallel implementation of some algorithms. Other topologies can be handled in a similar fashion.

2.3 Connections

The MAS defines, how the connections among all the building blocks should be arranged. Basically, each connection specifies the two endpoints: which building block–gate pair should be connected to which building block–interface pair. There are four cases of whether the connection targets from/to "inside" of the MAS or from/to "outside" of it, but only three out of them are interesting:

- inside to inside — both building blocks are inside the MAS being created;
- outisde to inside — the interface is inside the MAS but it is supposed to be connected to the gate from outside of the MAS being created;
- inside to outside — the gate is inside the MAS and it is supposed to target to some interface which is out of the MAS being created

2.4 Implementation

We will now describe the way how the foregoing concepts are implemented in bang[3] project.

Gate

The gates are implemented very strightforwardly. The `CLink` class is taken as the base class for the newly created `CGate` class. `CLink` is used in bang as a basic reference to an agent. It is used mostly by the `Sync` function which is designed for message passing[4]. `CLink` stores the information about the target — the unambiguous identification of the agent which it points to.

For our purposes this is insufficient: we also need to store the name of the interface which the particular connection targets to. The `CGate` class adds this property to the `CLink` class. The instance of `CGate` class can be created from any `CLink` instance providing the aditional information about the interface the gate is about to point to.

Interface

The basic means for accepting messages in bang is the concept of *triggers*. Each trigger is defined by two important parts: the message template and the chunk of code. The message template defines the constraints on the incomming message. If the incomming message holds the constraints, the given chunk of code is executed.

Whenever the `CGate` is used instead of `CLink` for sending the message, the information about the interface the gate targets to is added to the message as a metadata. An enhancement of the message template definition was added which makes it possible and easy to specify the interface in the trigger definition.

MAS

MAS is a common name for agents or suits of agents that fulfill the scheme definition, that can eventually manipulate their connections in order to incorporate themselves into another scheme.

In the case of the MAS consisting of a single agent, the situation is simple. Concerning sending messages, the agent is supposed use the `CGates` instead of `CLinks` for its communication. The agent is also supposed to understand the messages for manipulating the gates. On the other hand, when receiving messages, the agent is supposed to process the messages that come through its interfaces. We will now focus on the case of the suites of agents.

[3] www.cs.cas.cz/bang3

[4] To be precise, there are 3 basic functions for sending a message which differ on whether and how the reply is delivered to the sender. However, this is out of scope of this text. Only `Sync` is therefore used for simplicity in place, where all thre variants should have been mentioned.

The objective is to define the means for describing and maintaining them. We proposed and implemented an agent `AMas` that acts as a maintainer of one particular MAS. This agent is responsible for creating the MAS out of its description.

Agent `AMas` intensively uses so called Sub-Agent Modules (`SAM`s). The `SAM` is a concept which allows to create the modules that have *some* of the properties of the agents. Namely, it can create and send messages and it can specify the *triggers* (which basically means that it can accept and process incomming messages). However, the `SAM` cannot do it on its own — it always acts on the account of certain agent. The agents can *plug* the `SAM`s in. When the agent plugs the `SAM`, it adopts the triggers specified by the SAM and can take advantage of using its methods. The messages specified by the `SAM` to be sent appear as if they have been sent by the agent itself.

The description of the MAS contains two lists: the list of building blocks and the list of connections. There is more types of building blocks as well as there is more types of connections. There exists the `SAM` for each particular building block or connection. When constructing the MAS, the `AMas` agent reads the description (building blocks and connections) and cerates the relevant `SAM`s, and plugs them into itself.

According to their purpose, the following `SAM`s were implemented:

- `SMASCtor` — the base class SAM or all the building block descriptor classes mentioned later in this list. `SMASCtor` (and therefore each of its successors) keeps the link (`CLink`[5]) to the target agent. It also stores the `roleName` (a string which serves as a reference to the particular building block within the MAS).

  `SMASCtor` defines a virtual method (`incarnate`). This method is called by `AMas` when this agent/MAS should be created. `SMASCtor` also defines an auxiliary method `findAgent` serving for finding the agent of given `roleName` within the MAS, and another auxiliary method `createAgent` which creates new agent of given type. The standard mechanism of launching agents in Bang (agent `ALauncher`) is used here.

- `SMASFromTypeName` — building block SAM used for launching a new agent (i.e. not reusing an existing one) of the type that is given as a parameter.

- `SMASFromLink` — SAM for integrating an existing ("living") agent into the MAS. The link to the agent is given as a parameter.

- `SMASFromMAS` — SAM for integrating a "subMAS" into the MAS. `SMASFromMAS` is one of the ways how to get the MASes recursive. An "inner" MAS will be created according to the description (given as a parameter), and will be treated as a building block of the "outer" MAS.

- `SMASRing` — the token ring is created. Each node of the building block is a MAS of the same definition, which is given as a parameter.

  The specified MAS is created more times, and interconnected to form a token ring. The gate `ringnext` of each "clone" of the "inner MAS" is connected to the interface `ringnext` of the next one in the ring.

  Any message sent to this MAS as a whole is sent to the *first* member of the ring only. `SMASRing` provides also the means for sending message to the particular member of the ring, and for broadcasting a message to all the ring members.

  Example of uisng `SMASRing` is shown in the next section.

When agent `AMas` "incarnates" the MAS, first all building blocks are created/located. This is done by employing the virtual method `incarnate` of each particular building block. The `CLink` to each of them is stored in the corresponding SAM.

When links to all building blocks are avilable, `AMas` creates the connections among them. The connections are listed in the array which was mentioned earlier as the second part of the MAS definition.

The following SAMs for defining the interconnections among building blocks are avilable:

[5] We really keep a `CLink` here (and not `CGate`). We need a "pointer" to the whole inferior building block (agent/(sub)MAS) and not to its particular interface here...

- `SConnCtor` — the base class of all the connection descriptor's classes. The rest of classes in this list are derived from this one. `SConnCtor` declares the virtual method `establish`. The method is called when the scheme is built.

- `SConnInside` — describes the connection, where both source and receiver are within boundary of this MAS. The source is determined by name of building block (referring to the list of building blocks) and the name of gate (**G**). The destination is determined by name of building block and name of the interface (**I**). When establishing the connection, `AMas` finds the source agent, and instructs it to set its gate **G** to target to interface **I** of the target agent.

- `SConnIface` — describes the connection, that should act as a interface from "outside" into the MAS. Since we want the MAS to behave like an agent, which fulfills particular scheme definition, this gives the MAS the possibility to have its own interface. The `SConnIface` is given three parameters:

  - Name of the interface to be created. This interface belongs to the MAS as a whole and acts as a "fake" interface, which in fact is delivered to the interface described by the two following arguments.
  - Reference to the agent (let's call it **A**) which will be sent the messages comming through this interface
  - Name of the interface of agent **A**, which is to be used for delivering the messages

  To conclude, the definition of the interface of the MAS determines the particular agent–interface pair, into which the incomming messages should be delivered. There is established a trigger within the AMas agent, which triggers all messages comming through the given interface. This trigger forwards them to the building block's interface, that is specified.

- `SConnGate` — describes the connection which points out of the MAS and originates inside. Although this type of connection is described very similarly — by the agent–gate pair, which is the real originator, and by the name of the "fake" gate that formally belongs to the MAS as a whole — the implementation is quite different.

  In the "single agent" case we would want to have a gate, that stores the target–interface pair. We would want it to understand the set/get messages. Similarly, in our case we want the whole MAS to have a CGate that

  - is accessible by the building block that actually sends messages through
  - is able to understand the set/get messages

  `SConnGate` stores `CLink` to the agent who is going to be the originator of the messages and the name of its particular gate (both specified in the MAS definition).

The agent who puts all the above mentioned things together is the `AMas` agent.

3 Experiments

This section describes the experiments with automatical scheme generation using a genetic algorithm [1].

The training sets used for experiments represented various polynomials. The genetic algorithm was generating the schemes containing the following agents representing arithmetical operations: *Plus* (performs the addition on floats), *Mul* (performs the multiplication on floats), *Copy* (copies the only input (float) to two float outputs), *Round* (rounds the incoming float to the integer) and finally *Floatize* (converts the int input to the float).

The selected set of operators has the following features: it allows to build any polynomial with integer coefficients. The presence of the *Round* allows also another functions to be assembled. These functions are the 'polynomials with steps' that are caused by using the *Round* during the computation.

The only constant value that is provided is -1. All other integers must be computed from it using the other blocks. This makes it more difficult to achieve the function with higher coefficients.

We supply three operators for the genetic algorithm that would operate on graphs representing schemes: *random scheme creation*, *mutation* and *crossover*.

The aim of the first one is to create a random scheme. This operator is used when creating the first (random) generation. The diversity of the schemes that are generated is the most important feature the generated schemes should have. The 'quality' of the scheme (that means whether the scheme computes the desired function or not) is insignificant at that moment, it is a task of other parts of the genetic algorithm to assure this. The algorithm for random scheme creation works incrementally. In each step one building block is added to the scheme being created. In the beginning, the most emphasis is put on the randomness. Later the building blocks are selected more in fashion so it would create the scheme with the desired number and types of gates (so the process converges to the desired type of function).

The goal of the crossover operator is to create offsprings from two parents. The crossover operator proposed for scheme generation creates one offspring. The operator horizontally divides the mother and the father, takes the first part from father's scheme, and the second from mother's one. The crossover is illustrated in Fig. 3.1.

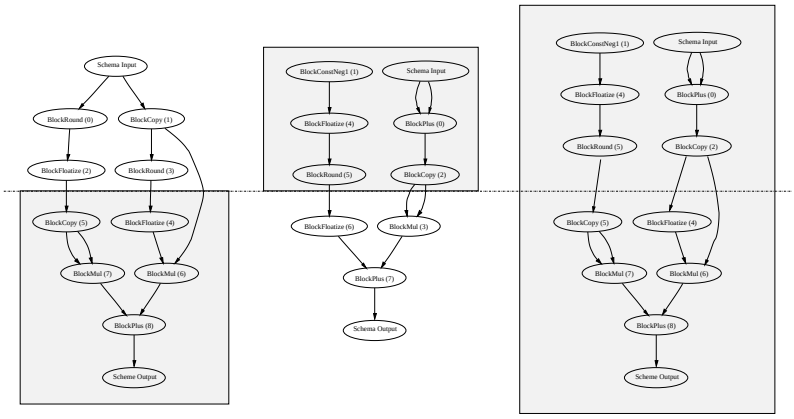

Figure 3.1: Crossover of two schemes. The mother and father are horizontally divided and the offspring becomes a mixture of both.

The mutation operator is very simple. It finds two links in the scheme (of the same type) and switches their destinations. The mutation operator is illustrated in Fig. 3.2.

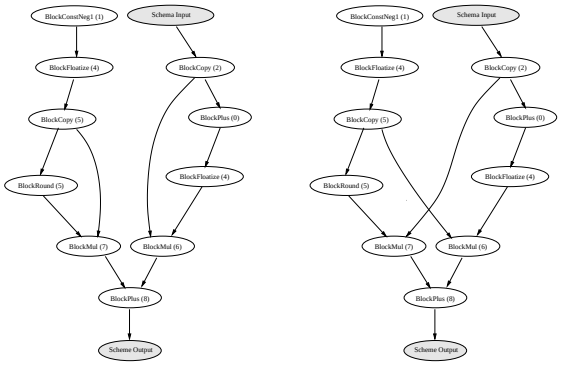

Figure 3.2: Mutation on scheme. The destination of two links are switched.

The aim of the experiments was to verify the possibilities of the scheme generation by genetic algorithms. The below mentioned examples were computed on 1.4GHz Pentium computers. The

computation is relatively time demanding. The duration of the experiment depended on many parameters. Generally, one generation took from seconds to minutes to be computed.

The results of the experiments depended on the complexity of the desired functions. The functions, that the genetic algorithm learned well and quite quickly were functions like $x^3 - x$ or x^2y^2. The learning of these functions took from tens to hundred generations, and the result scheme precisely computed the desired function.

Also more complicated functions were successfully evolved. The progress of evolving function $x^3 - 2x^2 - 3$ can be seen in the Fig. 3.3 and 3.4. Having in mind, that the only constant that can be used in the scheme is -1, we can see, that the scheme is quite big (comparing to the previous example where there was only approximately 5–10 building blocks) — see Fig. 3.5. It took much more time/generations to achieve the maximal fitness (namely 3000 generations) in this case.

On the other hand, learning of some functions remained in the local maxima, which was for example the case of the function $x^2 + y^2 + x$.

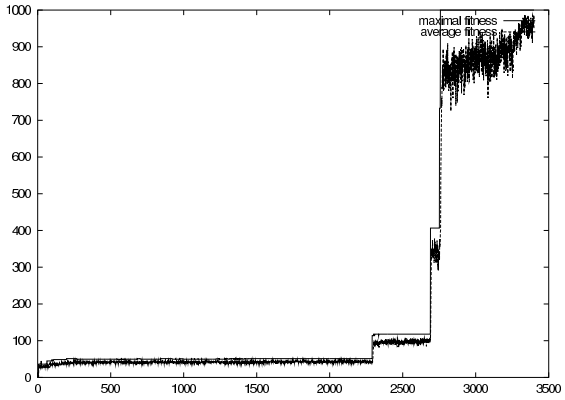

Figure 3.3: Function $x^3 - 2x^2 - 3$. The history of the maximal and average fitness

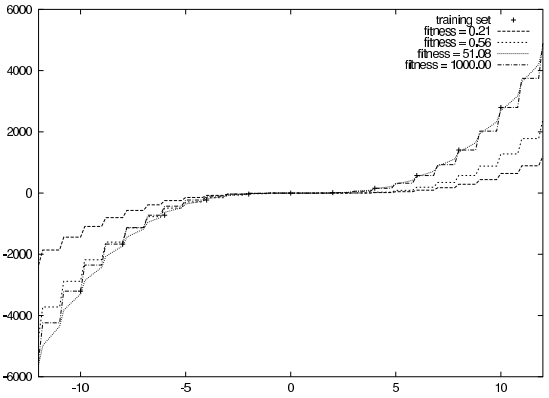

Figure 3.4: Function $x^3 - 2x^2 - 3$. The best schemes from generation 0, 5, 200 and 3000

4 Conclusion

We have introduced the notion of the schemes of agents in the multiagent systems. We have described the infrastructure for maintaining the schemes in the multiagent system bang. We have designed the system for description of such a schemes, and the instruments for building the schemes out of the agents and the other building blocks.

We have used an evolutionary algorithm for automatic creation of multiagent systems. Although the presented example limits to relatively simple agents for computing the arithmetical expressions,

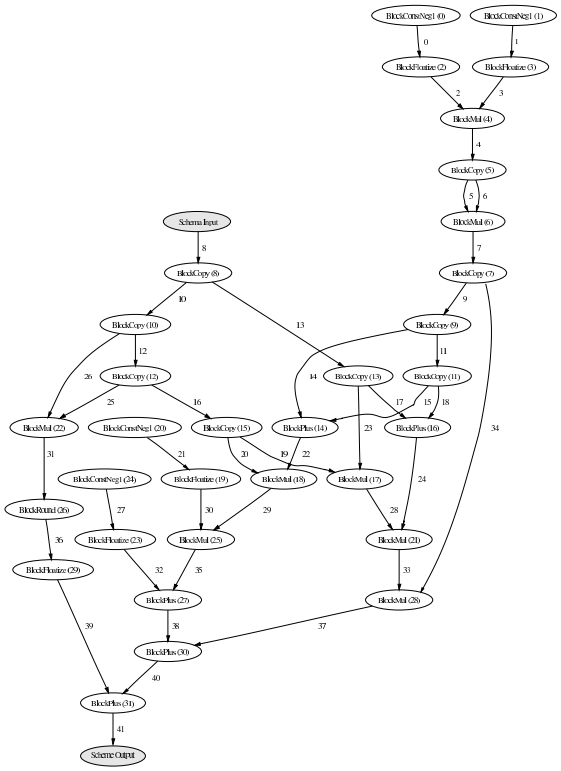

Figure 3.5: Function $x^3 - 2x^2 - 3$. The scheme with fitness 1000 (out of 1000), taken from 3000th generation.

we have demonstrated that it is possible to create the schemes of the multiagent systems in an autonomous, automatical way.

In the future work, we plan to focus on incorporating more complex agents into autonomously generated schemes.

Bibliography

[1] Gerd Beuster, Pavel Krušina, Roman Neruda, and Pavel Rydvan. Towards building computational agent schemes. In *Artificial neural Nets and Genetic Algorithms — Proceedings of the ICANNGA 2003*. Springer Wien, 2003.

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

[3] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman, 1999.

[4] D. B. Fogel. *Evolutionary Computation: The Fossil Record*. MIT-IEEE Press, 1998.

[5] J. Holland. *Adaptation In Natural and Artificial Systems*. MIT Press, reprinted edition, 1992.

[6] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.