Hybrid models as cooperating computational agents

Roman Neruda

Institute of Computer Science,
Academy of Sciences of the Czech Republic,
Prague, Czech Republic

roman@cs.cas.cz

UTIA, 31.10.2006

Outline

1. Introduction
2. Description of Computational Agents
   - Description of Agent and MAS
   - Implementation
3. Evolutionary algorithm
   - Description
   - Experiments
4. Autonomous Behaviour Support
   - Architecture description
   - Experiments
5. Conclusions

Computational intelligence

- Soft computing (L.Zadeh): creative fusion of artificial neural networks, evolutionary algorithms, fuzzy logic controllers, ...
- Benefits over individual methods.
- No one underlying theory.
- Importance of heuristics, experiments, practical skills.
- Combination with 'hard' (statistics, numerical analysis) and formal methods.

Goals

- To describe various computational methods as sets of several cooperating agents.
- *MAS scheme* is a concept for describing the relations within such a set of agents.
- 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.

Outline

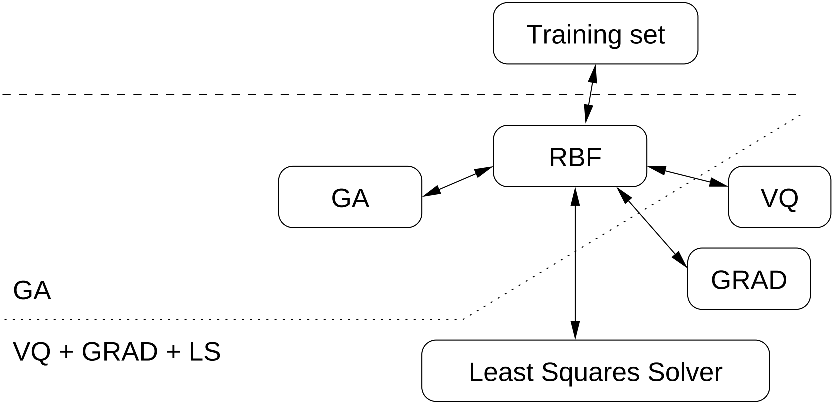
Bang as a middleware

- **support for agents life-cycle:** creation, migration, persistence,
- **communication:** message encoding, delivery
- **resource allocation:** memory, processor, disk
- **complexity analysis:** parallelization profiling
- **airport** on each computer, TCP/IP
- **agent granularity:** monolithic system / 1 or more threads per agent / processes
- **user interface**

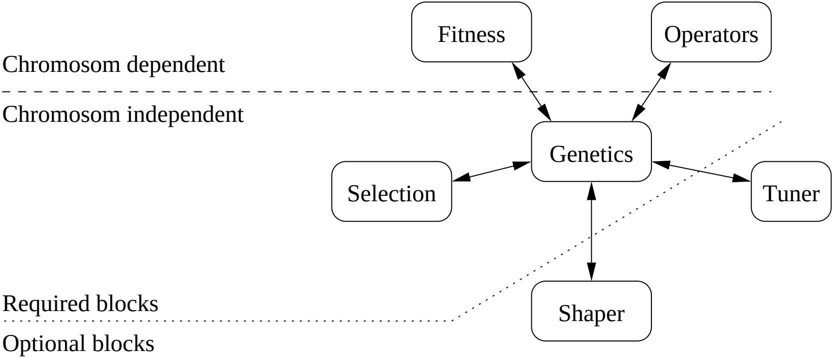
Agents in Bang

- **computational agents:** neural nets (MLP, RBF), GA suite, Kohonen maps, vector quantization, decission tree
- **computational helpers:** linear system solver, gradient descent optimization
- **task-related:** data source, task manager, file system wrapper
- **system:** launcher, yellow pages, ontology services, debugger, profiler
- **other:** MASman, console, GUI

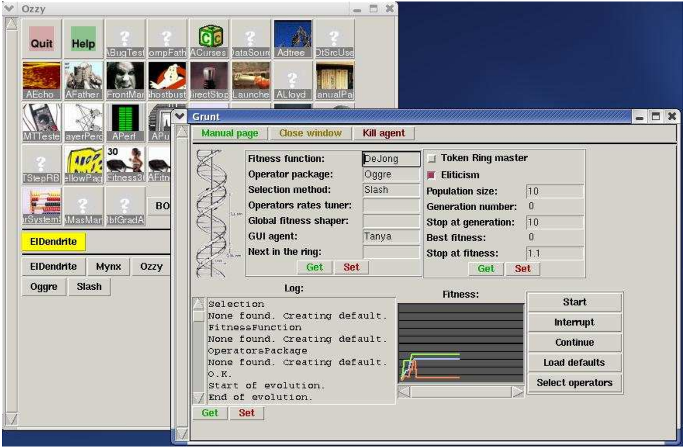
RBF as MAS


Training set
RBF
GA
VQ
GRAD
Least Squares Solver
GA
VQ + GRAD + LS

GA as MAS


Fitness
Operators
Chromosom dependent
Chromosom independent
Genetics
Selection
Tuner
Shaper
Required blocks
Optional blocks

GAs in action



Definitions: Communication

- **Message type:** identifies a category of messages that can be send to an agent in order to fulfill a specific task.
- **Interface:** the set of message types understood by a class of agents.
- **Gate:** a tuple consisting of a message type and a named link.
- **Connection:** a triple consisting of a sending agent, the sending agent's gate, and a receiving agent.

Definitions: Agents and MAS

- **Agent class:** defined by an interface, a set of message types, a set of gates, and a set of types.
- **Agent:** an instance of an agent class. It is defined by its name and its class.
- **Multi-Agent Systems (MAS):** consist of a set of agents, a set of connections between the agents, and the characteristics of the MAS.

Concepts and roles

Concepts	
mas(C)	C is a Multi-Agent System
class(C)	C is the name of an agent class
gate(C)	C is a gate
m_type(C)	C is a message type

Roles	
type(X,Y)	Class X is of type Y
has_gate(X,Y)	Class X has gate Y
gate_type(X,Y)	Gate X accepts messages of type Y
interface(X,Y)	Class X understands mess. of type Y
instance(X,Y)	Agent X is an instance of class Y
has_agent(X,Y)	Agent Y is part of MAS X

Computational agent

```
class(decision_tree)
type(decision_tree, computational_agent)
has_gate(decision_tree, data_in)
gate_type(data_in, training_data)
interface(decision_tree, control_messages)
...
```

Computational MAS

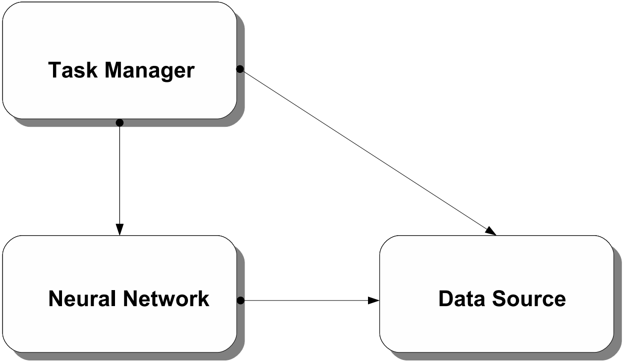
$$
\begin{aligned}
&\text{comp\_MAS(MAS)} \leftarrow \\
&\quad \text{type(CAC, computational\_agent)} \wedge \\
&\quad \text{instance(CA, CAC)} \wedge \\
&\quad \text{has\_agent(MAS, CA)} \wedge \\
&\quad \text{type(DSC, data\_source)} \wedge \\
&\quad \text{instance(DS, DSC)} \wedge \\
&\quad \text{has\_agent(MAS, DS)} \wedge \\
&\quad \text{connection(CA, DS, G)} \wedge \\
&\quad \text{type(TMC, task\_manager)} \wedge \\
&\quad \text{instance(TMC, TM)} \wedge \\
&\quad \text{has\_agent(MAS, TM)} \wedge \\
&\quad \text{connection(TM, CA, GC)} \wedge \\
&\quad \text{connection(TM, GC, GD)}
\end{aligned}
$$

Trusted MAS

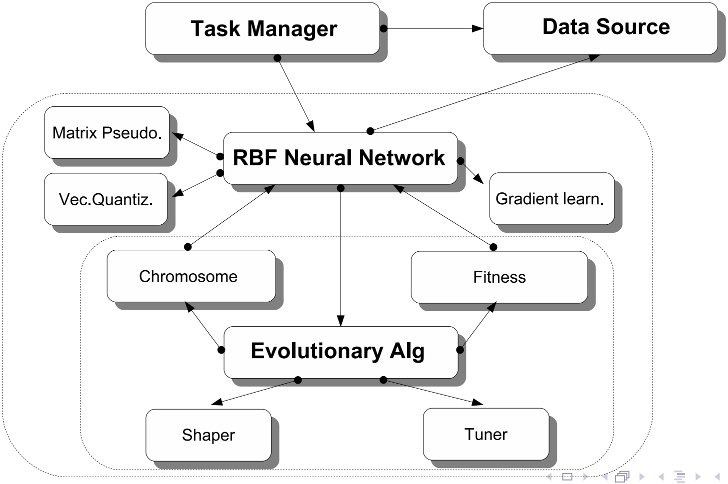
trusted_MAS(MAS) $\leftarrow$
  findall(X, has_agent(MAS,X), A))$\wedge$
  all_trusted(A)
all_trusted([]) $\leftarrow$ true
all_trusted([F|R]) $\leftarrow$
  instance(F,FC)$\wedge$
  type(FC, trusted) $\wedge$
  all_trusted([R])

MAS is trusted if all of its agents are instances of a "trusted" class. Prolog predicates `findall` (returns a list of all instances of a variable for which a predicate is true) and `all_trusted`.

Computational MAS



Computational MAS



Description of Agents

(implies iAgentStdIface (and
(some messagetype agentLifeManagement)
(all messagetype agentLifeManagement)))

(implies igToYellowPages (and
(some messagetype yellowPageRequest)
(all messagetype yellowPageRequest)))

(implies Father (and (some interface iAgentStdIface)
(all interface iAgentStdIface)
(some gate igToYellowPages)
(all gate igToYellowPages)))

...

Description of Agents

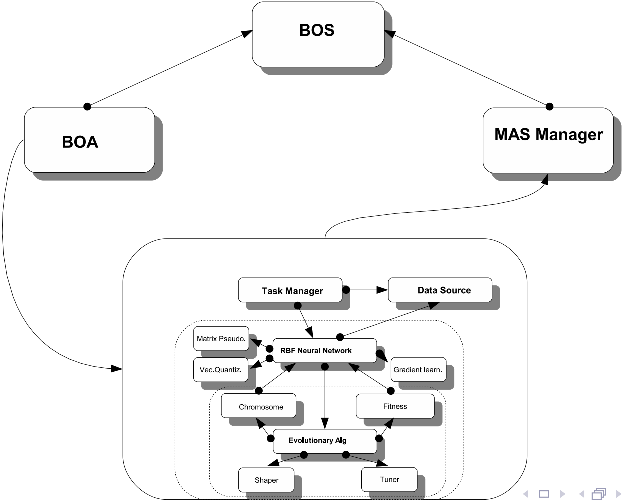
```
;;Decision Tree
(implies aDecisionTree (and Classifier
IterativeComputation
Father
classInBang))

;;Neural Networks
(implies NeuralNetwork Approximator)

;;RBF Network
(implies RBFNetworkAI (and NeuralNetwork
IterativeComputation
classInBang
SimpleTaskManager
Father
(some gate igSolveRepresentatives)
(some hide igCommonCompControl)
(all hide igCommonCompControl)
(some gate igSolveLinEqSystem)

(all gate (or igSolveRepresentatives igSolveLinEqSystem))))
```

Generation of MAS



Outline

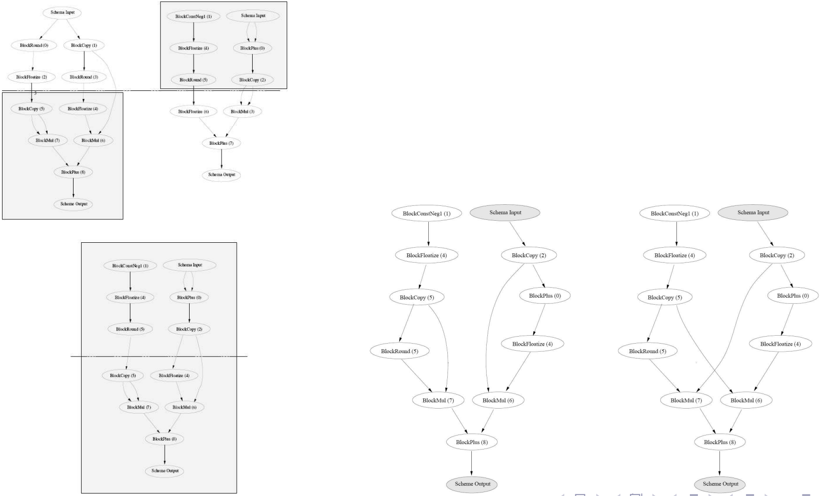
What it does

- EA operates on schemes definitions in order to find a suitable scheme solving a specified problem.
- Inputs:
  - number and the types of inputs and outputs of the scheme;
  - training set used to compute the fitness of a particular solution;
  - list of types of agents available for being used in the scheme.
- EA uses the agents logical description and reasoning component (described above) in order to produce only such schemes that satisfy given constrains.

How it works

- Random creation of population feasible schemes
- Evaluation of scheme:
  - Creating the MAS
  - Running it on the training set
  - Computing the fitness
- Creating new population by means of:
  - roulette-wheel selection
  - crossover of 2 schemes
  - 2 mutations of schemes (link swap, random node change)

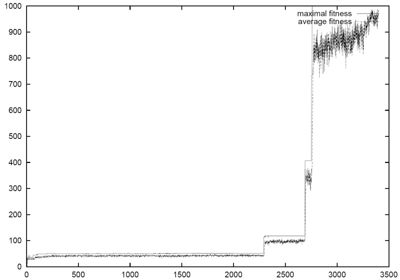
Crossover and Mutation

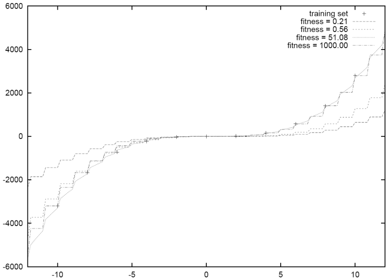


Symbolic regression

- Setup similar to J. Koza Genetic programming task
- Training set: 100 samples of a polynomial $x^3 - 2x^2 - 3$
- Agents: two families of agens working on INT and FLOAT values
- MUL, ADD, COPY, ROUND, FLOATIZE, ...
- 1 generation evaluation takes seconds on a 2GHz machine
- Several hundred/thousands generations needed to find a solution

Convergence





The best scheme from generation 3000

Outline

Adaptive Computational Agent

In order to act autonomously, an agent should be able to cope with three different kind of problems:

- cooperation of agents,
- computation processing support,
- optimization of the partner choice.

The architecture supports

- *reasoning*,
- *descriptions* of agents and tasks (ontologies),
- *monitoring* and *evaluation* of various parameters,
- *learning*.

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:

- deciding whether two agents are able to cooperate,
- evaluating the agents (according to reliability, speed, availability, etc.),
- reasoning about its own state of affairs (state of an agent, load, etc.),
- 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 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:

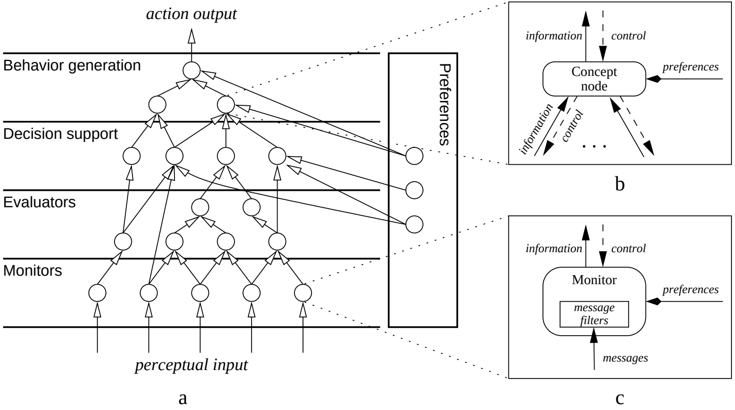
- learning (remembering) tasks the agent has computed in the past (we use the principles of case-based learning and reasoning to remember task cases),
- monitoring and evaluation of task parameters (duration, progress, count, etc.),
- evaluating tasks according to different criteria (duration, error, etc.).

Optimization of the partner choice

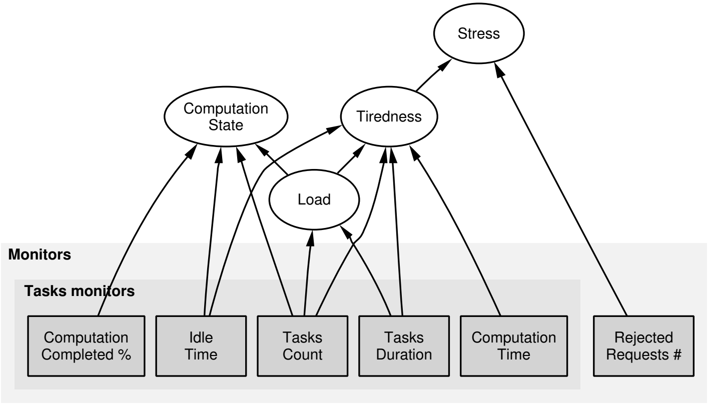
An intelligent agent should be able to distinguish good partners from unsuitable ones. The resulting subproblems follow:

- recognizing a suitable (admissible) partner for a particular task,
- increasing the quality of an evaluation with growing experience.

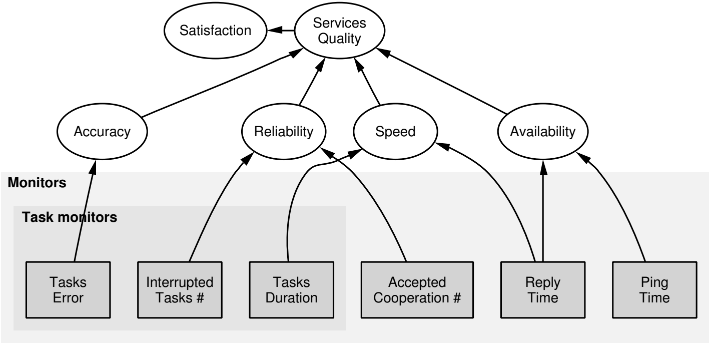
Layer architecture



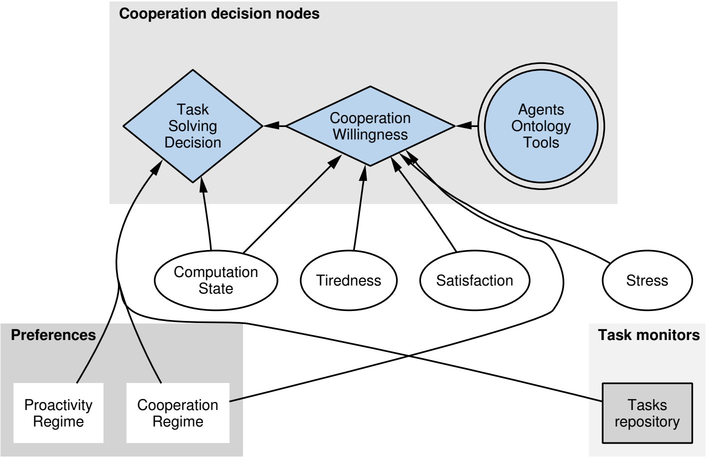
Modeling State of an Agent.



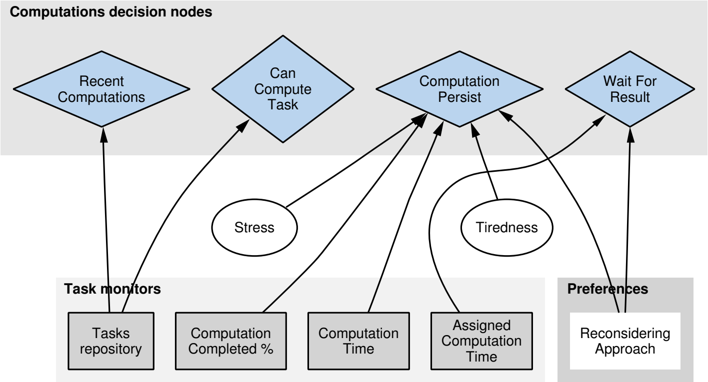
Measuring Quality of Services of Partners.



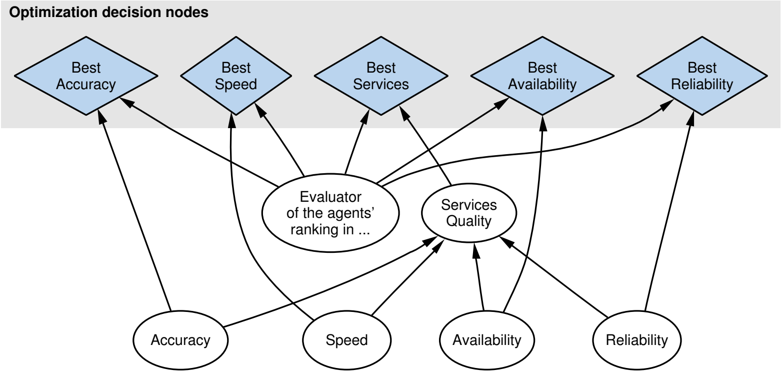
Support for Cooperation.



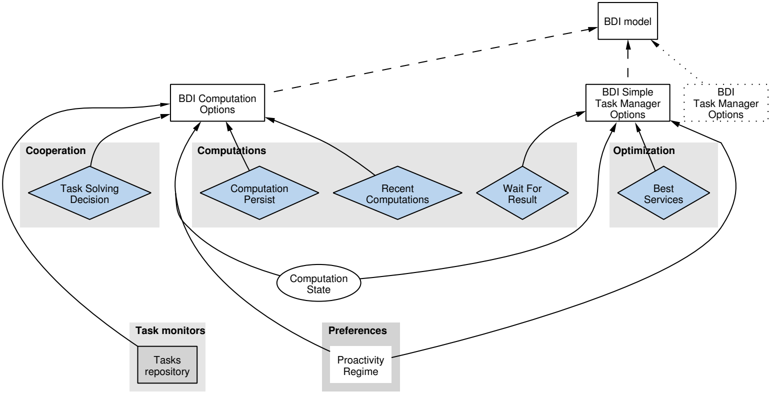
Computations Processing Support.



Optimization of Partner Choice.



BDI architecture within the Network of Concepts.



Experiments summary

- Testing on perceptron and local-unit neural networks, with two task managers, sets of (repeating) tasks.
- The architecture has (only) about 10% overhead.
- Optimization enhances efficiency of computational agents (w.r.t. different criteria).

Overhead of the architecture

	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

Optimization of partner choice

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

Optimization by Reusing

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

Outline

Conclusions

- Formal Description of computational agents and MAS ...
- ... by means of Description Logics and Prolog-like rules
- Application to automatic MAS scheme generation for simple problems
- Integration with Evolutionary algorihm on shemes
- Autonomous behavior support for computational agents

To do

- Dynamical aspects:
  - tasks description,
  - agents performance.
- User assistance, model verification.
- Use of First-Order Logics reasoning engine (KR-HYPER).
- Hybrid approaches, better combination with evolutionary search.
- Using the information from autonomous support in scheme evolution.