Toward Dynamic Generation of Computational Agents by Means of Logical Descriptions

Roman Neruda[1] and Gerd Beuster[2]

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou veží 2, 182 07 Prague 8, Czech Republic
[2] Institute of Informatics, University Koblenz-Landau
Universitätsstrasse 1, 56070 Koblenz, Germany
Email: roman@cs.cas.cz, gb@uni-koblenz.de

Abstract: A formalism for the logical description of computational agents and multi-agent systems is given. It is explained how it such a formal description can be used to configure and reason about multi-agent systems realizing computational intelligence models. A usage within a real software system *Bang 3* is demonstrated. A way to extend the system toward dynamic environments with migrating agents is discussed.
Keywords: Multi-agent systems, formal description, computational intelligence.

1. Introduction

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 [21]. 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 [10]. 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.

Bang 3 is a platform for the development of Multi-Agent Systems [16], [20]. Its main areas of application are computational intelligence methods (genetic algorithms, neural networks, fuzzy controllers) on single machines and clusters of workstations. 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 [6]. *Bang 3* — as a distributed multi-agent system — provides a support for an easy creation and execution of such hybrid AI models.

The system applications require a number of cooperating agents to fulfill a given task. So far, MAS are created and configured manually. In this paper, we introduce a logical reasoning component for *Bang 3*. With this component, *Bang 3* system configurations can be created automatically, triggered e.g. by an incoming task request, or user interaction.

Employing agent methodology for hybrid computational intelligence models represent a shift from one-time execution of a system on a given data toward pervasive computing environment, where persistent agents are distributed over a network of computing resources. These agents can migrate and take part in subsequent computations. They are also supposed to gather experience about their abilities and make use of them in order to better fulfill future tasks, find more suitable partners, etc. The formal mechanisms described in this paper are one of the possibilities how to describe and automate the composition of MAS based on given requirements and constrains. Our approach, although different in technical manners and specialized on a particular domain, tackles the same problem as the propitient MAS described in [2].

Let us finally mention that the logical description of computational MAS opens *Bang 3* for interaction with ontology based distributed knowledge systems like the Semantic Web and web services [14].

The description of *Bang 3* by formal logics enhances the construction, testing, and application of *Bang 3*-MAS in numerous ways:

- System Checking
  A common question in Multi-Agent System design is whether a setup has certain properties. By the use of formal descriptions of the agents involved in a MAS and their interactions, properties of the MAS can be (dis-)proved [18].
- System Generation
  Starting with a set of requirements, the reasoning component can be used to create a MAS from scratch. Partial reconfiguration is also possible, such as when a new computational agent appears and registers with the local directory services.
- Interactive System Generation
  The reasoning component can also be used to create agents in a semi-automated way by interaction. Here, the component can act as a helper application aiding a user in setting up MAS by searching for available agents, and making partial suggestions.
- Hybrid search methods
  The formal logical component can augment other search techniques, such as evolutionary algorithm that are already present in *Bang 3*and can be used for MAS configuration [5].
- Interaction with ontology based systems
  There is a growing interest in creating common logical frameworks (ontologies) that allow the interaction of independent, distributed knowledge based system. The most

prominent one is the Semantic Web, which attempts to augment the World Wide Web with ontological knowledge. Using formal logics and reasoning in *Bang 3* allows to open this world to *Bang 3*.

2. Description of MAS by means of Logics

In order to satisfy these requirements, the logical formalism must fulfill the following requirements:

1. It must be expressive enough to describe *Bang 3* MAS.
2. There must be efficient reasoning methods.
3. It should be suitable to describe ontologies
4. It should interface with other ontology based systems.

There is a lot of research in how to use formal logics to model ontologies. The goal of this research is to find logics that are both expressive enough to describe ontological concepts, and weak enough to allow efficient formal reasoning about ontologies.

The most natural approach to formalize ontologies is the use of First Order Predicate Logics (FOL). This approach is used by well known ontology description languages like Ontolingua [11] and KIF [13].

The disadvantage of FOL-based languages is the expressive power of FOL. FOL is undecidable [9], and there are no efficient reasoning procedures. Nowadays, the de facto standard for ontology description language for formal reasoning is the family of description logics. Description logics are equivalent to subsets of first order logic restricted to predicates of arity one and two [8]. They are known to be equivalent to modal logics [1].

Description logics are used in the Semantic Web, a project of the Internet standardization body W3C. The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers to deal with that information in a formal way. [4]. The Knowledge Grid project [24], [23] builds on top of the Semantic Web to create an intelligent environment allowing agents (both software and human) to share and manage knowledge. The objectives of the Knowledge Web are to support of team-work, problem-solving and decision making. Description logics is also the main topic of interest in other projects dealing with the standardization of inter-agent communications.

For the purpose of describing multi-agent systems, description logics are sometimes too weak. In these cases, we want to have a more expressive formalism. We decided to use Prolog-style logic programs for this. In the following chapters, we describe how both approaches can be combined together.

Description logics and Horn rules are orthogonal subsets of first order logic [8]. During the last years, a number of approaches to combine these two logical formalisms in one reasoning engine have been proposed. Most of these approaches use tableaux-style reasoners for description logics and combine them with Prolog-style Horn rules. In [15], Hustadt and Schmidt examined the relationship between resolution and tableaux proof systems for description logics. Baumgartner, Furbach and Thomas propose a combination of tableaux based reasoning and resolution on Horn logic [3]. Vellion [22] examines the relative complexity of SL-resolution and analytic tableau. The limits of combining description logics with horn rules are examined by Levy and Rousset [17]. Borgida [7] has shown that Description Logics and Horn rules are orthogonal subsets of first oder logic.

3. Computational Agents

An *agent* is an entity that has some form of perception of its environment, can act, and can communicate with other agents. It has specific skills and tries to achieve goals. A *Multi-Agent System (MAS)* is an assemble of interacting agents in a common environment [12].

In order to use automatic reasoning on a MAS, the MAS must be described in formal logics. For the *Bang 3* system, we define a formal description for the static characteristics of the agents, and their communication channels. We do not model dynamic aspects of the system yet.

Bang 3 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:

1. Open the data source located at XYZ,
2. Randomize the order of the data items,
3. Set the cursor to the first item,
4. Send next item.

These messages belong to a common category: Messages requesting input data from a data source. In order to abstract from the actual messages, we subsume all these messages under a *message type* when describing an agent in formal logics.

DEFINITION 1. *A* message type *identifies a category of messages that can be send to an agent in order to fulfill a specific task. We refer to message types by unique identifiers.*

The set of message types understood by an agent is called its *interface*. For outgoing messages, each link of an agent is associated with a message type. Via this link, only messages of the given type are sent. We call a link with its associated message type a *gate*.

DEFINITION 2. *An interface is the set of message types understood by a class of agents.*

DEFINITION 3. *A gate is a tuple consisting of a message type and a named link.*

Now it is easy to define if two agents can be connected: Agent A can be connected to agent B via gate G if the message type of G is in the list of interfaces of agent B. Note that one output gate sends messages of one type only, whereas one agent can receive different types of messages. This is a very natural concept: When an agent sends a message to some other agent via a gate, it assigns a specific role to the other agent, e.g. being a supplier of training data. On the receiving side, the receiving agent usually should understand a number of different types of messages, because it may have different roles for different agents.

DEFINITION 4. *A connection is described by a triple consisting of a sending agent, the sending agent's gate, and a receiving agent.*

Next we define *agents* and *agent classes*. *Bang 3* is object oriented. Agents are created by generating instances of classes. An agent derives all its characteristics from its class definition. Additionally, an agent has a name to identify it. The static aspects of an agent class are described by the interface of the agent class (the messages understood by the agents of this class), the gates of the agent (the messages send by agents of this class), and the type(s) of the agent class. Types are nominal identifiers for characteristics of an agent. The types used to describe the characteristics of the agents should be ontological sound.

Table 1. Concepts used to describe multi-agent system.

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

Table 2. Roles used to describe multi-agent system.

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

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)

Fig. 1. Example agent class definition.

DEFINITION 5. *An agent class is defined by an interface, a set of message types, a set of gates, and a set of types.*

DEFINITION 6. *An agent is an instance of an agent class. It is defined by its name and its class.*

4. Describing multi-agent systems

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 an empty MAS, or a manually constructed partial MAS.

DEFINITION 7. Multi-Agent Systems (MAS) *consist of a set of agents, a set of connections between the agents, and the characteristics of the MAS.*

Description logics know concepts (unary predicates) and roles (binary predicates). In order to describe agents and Multi-Agent Systems in description logics, the definitions 1 to 7 are mapped onto description logic concepts and roles as shown in Table 1 and Table 2.

An example agent class description is given in figure 1. It defines the agent class "decision_tree". This agent class accepts messages of type "control_message". It has one gate called "data_in" for data agent and emits messages of type "training_data".

In the same way, A-Box instances of agent classes are defined:

$$instance(decision\_tree, dt\_instance)$$

An agent is assigned to a MAS via role "has_agent". In the following example, we define "dt_instance" as belonging to MAS "my_mas":

$$has\_agent(my\_mas, dt\_instance)$$

Since connections are relations between three elements, a sending agent, a sending agent's gate, and a receiving agent, we can not formulate this relationship in traditional description logics. It would be possible to circumvent the problem by splitting the triple into two relationships, but this would be counter-intuitive to our goal of defining MAS in an ontological sound way. Connections between agents are relationships of arity three: Two agents are combined via a gate. Therefore, we do not use description logics, but traditional logic programs in Prolog notation to define connections:

$$connection(dt\_instance, other\_agent, gate)$$

Constraints on MAS can be described in Description Logics, in Prolog clauses, or in a combination of both. As an example, the following concept description requires the MAS "dt_MAS" to contain a decision tree agent:

$$dt\_MAS \sqsupseteq mas \sqcap has\_agent.(\exists instance.decision\_tree)$$

An essential requirement for a MAS is that agents are connected in a sane way: An agent should only connect to agents that understand its messages. According to definition 4, a connection is possible if the message type of the sending agent's output gate matches a message type of the receiving agent s interface. With the logical concepts and descriptions given in this section, this constraint can be formulated as a Prolog style horn rule. If we are only interested in checking if a connection satisfies this property, the rule is very simple:

connection(S,R,G) $\leftarrow$
instance(R, RC) $\wedge$
instance(S, SC) $\wedge$
interface(RC, MT)$\wedge$
has_gate(SC, G) $\wedge$
gate_type(G, MT)

The first two lines of the rule body determine the classes RC and SC of the sending agent S and the receiving agent R. The third line instantiates MT with a message type understood by RC. The fourth line instantiates G with a gate of class SC. The last line assures that gate G matches message type MT.

The following paragraphs show two examples for logical descriptions of MAS. It should be noted that these MAS types can be combined, i.e. it is possible to query for trusted, computational MAS.

Computational MAS: A computational MAS can be defined as a MAS with a task manager, a computational agent and a data source agent which are interconnected.

comp_MAS(MAS) ←
    type(CAC, computational_agent)∧
    instance(CA, CAC)∧
    has_agent(MAS, CA)∧
    type(DSC, data_source)∧
    instance(DS, DSC)∧
    has_agent(MAS, DS)∧
    connection(CA, DS, G)∧
    type(TMC, task_manager)∧
    instance(TMC, TM)∧
    has_agent(MAS, TM)∧
    connection(TM, CA, GC)∧
    connection(TM, DS, GD)

Trusted MAS: We define that an MAS is trusted if all of its agents are instances of a "trusted" class. This examples uses the Prolog predicate `findall`. `findall` returns a list of all instances of a variable for which a predicate is true. In the definition of predicate `all_trusted` the usual Prolog syntax for recursive definitions is used.

trusted_MAS(MAS) ←
    findall(X, has_agent(MAS,X), A))∧
    all_trusted(A)
all_trusted([]) ← true
all_trusted([F|R]) ←
    instance(F,FC)∧
    type(FC, trusted) ∧
    all_trusted([R])

5. Implementation

The above described concepts and algorithms are implemented within the *Bang 3* software system as the BOA agent. This agent works with ontological description files of the three kinds:

- the Description Logics description of agent hierarchies, their gates, interfaces and message types,
- the Prolog clauses describing more complicated properties and concepts, such as the form of computational MAS, or the notion of trust,
- the time-dependent information about available agents gathered at the time of particular task from directory services agent.

5.1 Computational multi-agent systems

In this section we give examples of a MAS scheme describing the computational MAS definition from section 4. A typical computational MAS configuration is shown on figure 2. There are two more complicated computational agents, the RBF neural network (RBF) and the Evolutionary algorithm (EA) agent, that cooperate with each other within a computational MAS. Each of these two agents can itself be seen as a MAS employing several simpler agents to solve a given task. In the case of the RBF network, typically an unsupervised learning (vector quantization), and a supervised learning (gradient, matrix inverse) agent is needed. The evolutionary algorithm agent makes use of fitness (shaper) and probabilities manager (tuner). The cooperation of RBF and EA is more intricate and takes place via the fitness and chromosome agents.

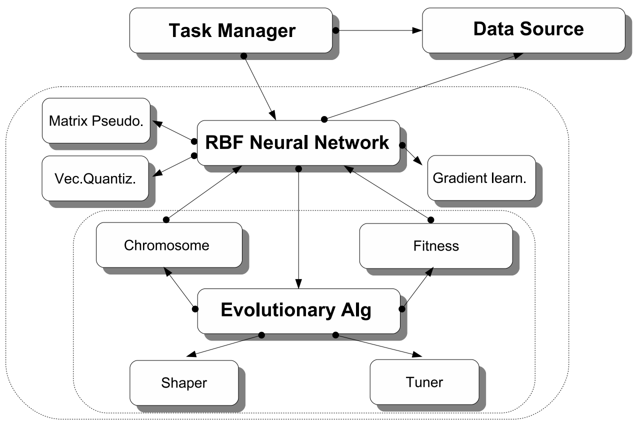

Fig. 2. Example of a more complicated computational MAS consisting of a Task Manager agent, Data Source agent, and a suite of cooperating computational agents (an RBF network agent and Evolutionary algorithm agent with necessary additional agents).

5.2 MAS descriptions

Descriptions of the above shown — and similar — multi-agent systems are generated by the BOA agent in a formal description language. This description is then sent to the MAS manager agent, which is able to take care of physical creation of the whole system. This includes creating suitable agents, linking their gates and interfaces, sending them appropriate initialization messages, etc. There are more possibilities during the creation of agents, it is possible to create new ones by constructing them. Other way is to try to reuse existing agents that are free at the moment, and yet another possibility is to find suitable agents by means of ontology services. All three ways of incorporating agents into schemes are accounted for in the description formalism. The creation of the computational MAS is typically followed by an (automated) trial and evaluation on a particular data set.

Another way of BOA work, which is currently being developed, is an integration with GUI MAS designer, where BOA invalidates connections that are not correct, and suggests suitable partners for a connection. This way of work considers a graphical tool which assists the user to design a valid MAS scheme in a WYSIWYG manner. Since the queries for the validity of connections and partners are rather simple, the time requirements are sensible considering the real-time performance.

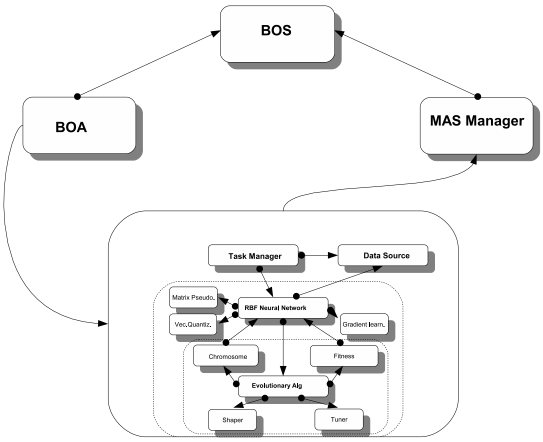

Fig. 3. The BOA agent generates a MAS configuration description and sends it to the MAS manager agent, which takes care of MAS creation and run. They both query the BOS ontology services agent.

Figure 4 demonstrates the above described ideas on the actual implementation of the agents hierarchy description in the RACER Lisp-like syntax. For the sake of simplicity, only the Decision Tree and RBF Neural Network are shown with several intermediate concepts missing. The complete description is included in [19].

```
(implies iAgentStdIface (and
         (some message_type agentLifeManagement)
     (all message_type agentLifeManagement)))

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

(implies Father (and (some interface iAgentStdIface)
     (all interface iAgentStdIface)
     (some gate igToYellowPages)
     (all gate igToYellowPages)))
...
;;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))
   (some interface igRunNetworkDemo)
   (all interface igRunNetworkDemo)))
```

Fig. 4. Example of agent ontology description in the RACER Lisp-like formalism.

5.3 Dynamic system configuration

So far, the system is used mainly in a static environment where the goal is to create a suitable MAS in order to fulfill a specified task, and execute it. This can be either a one-shot behavior, or a repeated task, such as in the case of evolutionary algorithm used for MAS optimization (cf. Conclusion). However, the system already contains two types of information:

- the static ontology descriptions, and possible more complicated constrains,
- and the dynamic information about currently available agents.

Thus, the BOA agent also keeps track of two types of data: the static and dynamic information. While the static information is typically downloaded once at BOA startup, the dynamic information about available agents and their types is retrieved from the directory services agents every time there is a new task to be solved. Thus, the BOA can work with all the agents currently available in the system (including the ones that migrated in recently), provided they register with the directory services agent.

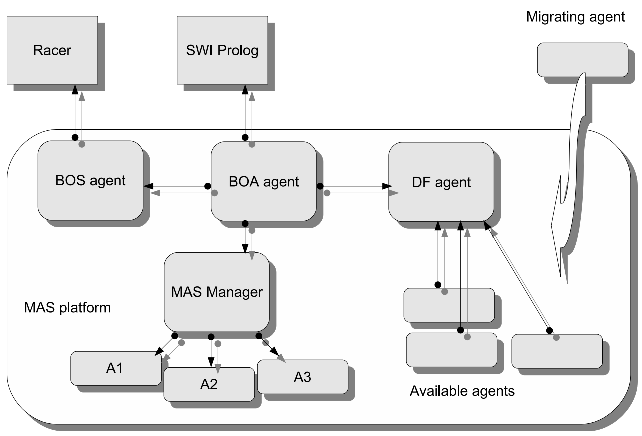

Fig. 5. The BOA agent refreshes its dynamic information about available agents from DF agent before solving every task, thus taking even currently arrived agents into account.

6. Conclusion

We have shown how formal logics can be used to describe computational MAS. We presented a logical formalism for the description of MAS. In this, we combined Description Logics with traditional Prolog rules. The system we implemented allows the practical application of these technologies. We have demonstrated how this approach works in practice within the hybrid computational environment *Bang 3*.

So far, we have achieved a partial support for dynamic environments with migrating agents. The constrain parts and ontological description of agents and their properties is still static, but the MAS creation reflects dynamic changes in the system. Further research will be put in the development of formal descriptions of dynamic aspects of MAS. In particular, this means to work with ontological description of tasks and to gather knowledge about computational agents performance. Currently within *Bang 3*, there is a BDI-based mechanism that supports local decisions of a computational agent based on its previous experience. This will blend smoothly with our approach, which in turn allows to provide more suitable MAS solutions. In particular, if there are more agents satisfying the constrains, we will be able to sort them according to their past performance in the required context. Thus, better partners for an agent can be supplied. Further in the future we plan to employ proactive mechanisms for an agent (again BDI-based), which will be allowed to improve its knowledge in its free time, such as trying to solve benchmark tasks and recording the results.

The hybrid character of the system, with both a logical component and soft computing agents, also makes it interesting to combine these two approaches in one reasoning component. In order to automatically come up with feasible hybrid solutions for specific problems, we plan to combine two orthogonal approaches: a soft computing evolutionary algorithm with a formal ontology-based model. So far, in [5] we have tried the isolated evolutionary approach, and the results, although satisfiable, are difficult to scale up to larger configurations. We expect synergy effects from using formal logics to aid evolutionary algorithms and vice versa.

Acknowledgments

This work has been partially supported by the the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) ”Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realization”.

References

[1] F Baader, Logic-based knowledge representation, In Artificial Intelligence Today, Recent Trends and Developments, M J Wooldrige and M Veloso, Eds. Springer, 1999, pp. 13–41.

[2] M Bakhouya and J Gaber, Self-organizing approach for emergent multi-agent structures. Workshop on Complexity through Development and Self-Organizing Representations at GECCO’06 Genetic and Evolutionary Computation Conference 2006, ACM Press.

[3] P Baumgartner, U Furbach and B Thomas, Model-based deduction for knowledge representation. Proceedings of the International Workshop on the Semantic Web Hawaii, USA, 2002.

[4] T Berners-Lee, J Hendler and O Lassila, The semantic web. Scientific American, 2001.

[5] G Beuster, P Krušina, R Neruda and P Rydvan, Towards building computational agent schemes. Artificial neural Nets and Genetic Algorithms — Proceedings of the ICANNGA 2003 2003, Springer Wien.

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

[7] A Borgida, On the relationship between description logic and predicate logic. CIKM, 1994, pp. 219–225.

[8] A Borgida, On the relative expressiveness of description logics and predicate logics. Artificial Intelligence, Vol. 82, No. 1–2, 1996, pp. 353–367.

[9] M Davis, Ed., The Undecidable—Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions, Raven Press, 1965.

[10] J E Doran, S Franklin, N R Jennings and T J Norman, On cooperation in multi-agent systems. The Knowledge Engineering Review, Vol. 12, No. 3, 1997, pp. 309–314.

[11] A Farquhar, R Fikes and J Rice, Tools for assembling modular ontologies in ontolingua, Tech. rep., Stanford Knowledge Systems Laboratory, 1997.

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

[13] M R Genesreth and R E Fikes, Knowledge interchange format, version 2.2, Tech. rep., Computer Science Department, Stanford University, 1992.

[14] J Hendler, Agents and the semantic web. IEEE Intelligent Systems, Vol. 16, No. 2, 2001, pp. 30–37.

[15] U Hustadt and R A Schmidt, On the relation of resolution and tableaux proof system for description logics. Proceedings of the 16th Internatoinal joint Conference on Artificial Intelligence IJCAI’99 Stockholm, Sweden, 1999, D Thomas, Ed., Vol. 1, Morgan Kaufmann, pp. 110–115.

[16] P Krušina, R Neruda and Z Petrova, More autonomous hybrid models in bang. International Conference on Computational Science (2) 2001, Springer Verlag, pp. 935–942.

[17] A Y Levy and M C Rousset, The limits of combining recursive horn rules with description logics. Proceedings of the Thirteenth National Conference on Artificial Intelligence Portland, OR, 1996.

[18] R Meolic, T Kapus and Z Brezocnik, Model checking: A formal method for safety assurance of logistic systems. 2nd Congress Transport – Traffic – Logistics Portoroz, Slovenia, 2000, pp. 355–358.

[19] R Neruda, et al., Bang 3 multi-agent system documentation, 2006, http://bang.sf.org.

[20] R Neruda, P Krušina, P Kudova and G Beuster, Bang 3: A computational multi-agent system. Proceedings of the 2004 WI-IAT’04 Conference 2004, IEEE Computer Society Press.

[21] H S Nwana, Software agents: An overview. Knowledge Engineering Review, Vol. 11, No. 2, 1995, pp. 205–244.

[22] A Vellino, The relative complexity of sl-resolution and analytical tableau. Studia Logica, Vol. 52, No. 2, 1993, pp. 323–337, Kluewer.

[23] H Zhuge, Semantic space grid: Model, method and platform. Concurrency and Computation:Practice and Experience, 2004, in press.

[24] H Zhuge, Semantics, resource and grid. Future Generation Computer Systems, Vol. 20, No. 1, 2004, pp. 1–5.

Authors Bios

Roman Neruda received his PhD in theoretical computer science from Academy of Sciences of the Czech Republic in 1998. He is a research fellow at the Institute of Computer Science of the Academy of Sciences of the Czech Republic, and a lecturer at Faculty of Mathematics and Physics, Charles University in Prague. His research interests include intelligent agents and multi-agent systems and computational intelligence, namely hybrid models. He has published more than 50 papers in international journals and proceedings of international conferences, and co-authored or co-edited four books.

Gerd Beuster is a PhD student and a research assistant at the Institute of Informatics of the Koblenz-Landau University in Germany. His research interests include formal description and formal verification of software systems, and computational intelligence. He has published about a dozen of papers in international journals and proceedings of international conferences.