DEVELOPING AGENT INTERACTION PROTOCOLS WITH PRALU

Dmitry Cheremisinov, Liudmila Cheremisinova

Abstract. The purpose of the paper is to explore the possibility of applying existing formal theories of description and design of distributed and concurrent systems to interaction protocols for real-time multi-agent systems. In particular it is shown how the language PRALU, proposed for description of parallel logical control algorithms and rooted in the Petri net formalism, can be used for the modeling of complex concurrent conversations between agents in a multi-agent system. It is demonstrated with a known example of English auction on how to specify an agent interaction protocol using considered means.

Keywords: multi-agent system, interaction protocol, parallel control algorithm

ACM Classification Keywords: I.2.11 [Computer Applications]; Distributed Artificial Intelligence, Multiagent systems; D.3.3 [Programming Languages]: Language Constructs and Features – Control structures, Concurrent programming structures

Introduction

Agents are becoming one of the most important topics in distributed and autonomous decentralized systems, and there are increasing attempts to use agent technologies to develop large-scale commercial and industrial software systems. Over the last decade, the specification, design, verification and application of a particular type of agents, called BDI (belief, desire, intention) agents [1], have received a great deal of attention. BDI agents are systems that are situated in changing environment receive continuous perceptual input and take actions to affect their environment based on their internal state. In particular, there are many efforts aimed at developing agent-oriented designs that are typically structured as multi-agent systems (MASs). MAS is a computational system in which two or more agents interact or work together to perform a set of tasks or to achieve a set of goals [2]. Agents of a MAS interact with others toward their common specific objective or individual benefit. Agent interactions are established through exchanging messages that specify the desired performatives of other agents (such as notice, request) and declarative representations of the content of messages.

MAS is usually specified as a concurrent system based on the notion of autonomous, reactive and internally-motivated agents acting in a decentralized environment. One key reason of the growth of interest in MAS is that the idea of an agent as an autonomous system, capable of interacting with other agents in order to satisfy its design objectives, is a naturally appealing one for software designers. Agents are used in an increasingly wide variety of applications such as distributed and autonomous decentralized systems. The complexity of MASs suggests a pressing need for system modeling techniques to support reliable, maintainable and extensible design. Although there are many efforts aimed at developing such MASs, there is sparse research on formal specification and design of such systems.

Agent system can operate if the agents are able to exchange information in the form of messages and if they have a common understanding of the possible types of messages that are connected with message “content”. This shared understanding is referred to as ontology [3]. A great deal of agent-based research has devoted to the development of techniques for representing ontology’s. Multi-agent conversations are built upon two components: agent communication language and interaction protocol. There are a number of agent communication languages, such as Knowledge Query and Manipulation Language (KQML) [4] and the Foundation for Intelligent Physical Agents (FIPA) ACL [5] and others designed for special purposes and that are like mentioned ones. These agent communication languages specify a domain specific vocabulary (ontology) and the individual messages that can be exchanged between agents.

Interaction protocols [5] specify the sequences in which these messages should be arranged in agent interactions. A group of rational agents complies with an interaction protocol in order to engage in task-oriented sequences of message exchange. Thus, when an agent sends a message, it can expect a receiver's response to

be among a set of messages indicated by the protocol and the interaction history. With a common interpretation of the protocol, each member of the group can also use the rules of the interaction in order to satisfy its own goals. In other words protocol constrains the sequences of allowed messages for each agent at any stage during a communicative interaction (dialogue), i.e. it describes some standard pattern messages exchanged between agents need to follow. Protocol plays a central role in agent communication. It specifies the rules of interaction between communicating agents and the first thing should be done by the designer developing any particular real-time system is to impose interaction protocol.

This paper explores the possibility of applying existing formal theories of description of distributed and concurrent systems to interaction protocols for real-time multi-agent systems. In particular it is shown how the language PRALU [6, 7], proposed for description of parallel logical control algorithms and rooted in the Petri net formalism, can be used to describe agent interaction protocols. The described approach can be used for the modeling of complex, concurrent conversations between agents in a multi-agent system. It can be used to define protocols for complex conversations composed of a great number of simpler conversations. With the language PRALU it is possible to express graphically the concurrent characteristics of a conversation, to capture the state of a complex conversation during runtime, and to reuse described conversation structure for processing multiple concurrent messages. It is demonstrated with a known example of English auction [5] on how to specify an agent interaction protocol using considered means. Finally, using PRALU language we can verify some key behavioral properties of our protocol description that is facilitated by the use of existing software for the language PRALU [6, 7, 8].

Agent Interaction Protocols

The application domains of MASs are getting more and more complex. Firstly, many current application domains of MASs require agents to work in changing environment (or world) that acts on or is acted on by the system. A closed system is one that has no environment; it is completely self-contained in contrast to an open (uncertain) system, which interacts with its environment. Any real system is open. The MAS must decide what to do and develop a strategy in order to achieve its assigned goals. For this, the MAS must have a representation or a model of the environment in which it evolves. The environment is composed of situations. A situation is the complete state of the world at an instant of time.

The application of multi-agent systems to real-time environments can provide new solutions to very complex and restrictive systems such as real-time systems. A suitable method for real-time multi-agent system development must take into account the intrinsic characteristics of systems of this type. As a rule they are distributed, concurrent systems with adaptive and intelligent behaviour. For agent-based systems to operate effectively, they must understand messages that have a common ontology underlying them. Understanding messages that refer to ontology can require a considerable amount of reasoning on the part of the agents, and this can affect system performance.

BDI agents are systems that are situated in a changing environment, receive continuous perceptual input, and take actions to affect their environment, all based on their internal mental state. Within the BDI architecture agents are associated with beliefs (typically about the environment and other agents), desires or goals to achieve, and intentions or plans to act upon to achieve its desires. In practical terms, beliefs represent the information an agent has about the state of the environment. It is updated appropriately after each action. The desires denote the objectives to be accomplished, including what priorities are associated with the various objectives. Intentions reflect the actions that must be fulfilled to achieve the goal (the rules to be fired).

To reduce the search space of possible responses to agent messages, interaction protocols can be employed. They specify a limited range of responses that are appropriate to specific message types for a given protocol. When an agent is involved in a conversation that uses an interaction protocol, it maintains a representation of the protocol that keeps track of the current state of the conversation. After a message is received or sent, it updates the state of the conversation in this representation.

By the very nature of protocols as public conventions, it is desirable to use a formal language to represent them. When agents are involved in interactions where no concurrency is allowed, conversation protocols are traditionally specified as deterministic finite automata (DFA) of which there are numerous examples in the literature. DFA consists of a set of states, an input alphabet and a transition function, which maps every pair of state and input to next state. In the context of interaction protocols [9], the transitions specify the communicative

actions to be used by the various agents involved in a conversation. A protocol based on such a DFA representation determines a class of well-formed conversations. Conversations that are defined in this way have a fixed structure that can be laid down using some kind of graphical representation.

Protocols can be represented as well in a variety of other ways. The simplest is a message flow diagram, as used by FIPA [5]. More complex protocols will be better represented using a UML sequence (Uniffied Modelling Language) [10] and AUML [11], interaction diagram, statechart [12] and Colored Petri Net (CPN) [13]. UML is one of the currently most popular graphical design languages that are de facto standard for the description of software systems. AUML extends UML sequence diagrams to represent asynchronous exchange of messages between agents. The advantage of AUML is its visual representation. Statecharts [12] are an extension of conventional DFAs. However, expressing protocols of realistic complexity using statecharts or AUML requires substantial efforts for developing, debugging and understanding.

A CPN model of a system describes the states, which the system may be in, and the transitions between these states. CPNs provide an appropriate mathematical formalism for the description, construction and analysis of distributed and concurrent systems. CPNs can express a great range of interactions in graphical representations and well-defined semantics, and allow formal analysis and transformations [14]. By using CPNs, an agent interaction protocol can be modeled as a net of components, which are carriers of the protocol structure. Using CPNs to model agent interaction protocol, the states of an agent interaction are represented by CPN places. Each place has an associated type determining the kind of data that the place may contain. Data exchanged between agents are represented by tokens, and the colors of tokens indicate the data value of the tokens. The interaction policies of a protocol are carried by CPN transitions and their associated arcs. A transition is enabled if all of its input places have tokens, and the colors of these tokens can satisfy constraints that are specified on the arcs. A transition can be *fire*d, which means the actions of this transition can occur, when this transition is enabled. When a transition occurs, it consumes all the input tokens as computing parameters, conducts conversation policy and adds new tokens into all of its output places. After a transition occurs, the state (marking) of a protocol has been changed and a protocol will be in terminal state when there is no enabled or fired transition.

There are a number of works of using Petri Nets or CPNs to model agent interaction protocols [15, 16], there have been also some works on the investigation of protocols' flexibility, robustness and extensibility [17]. It is becoming consensus [14] that a CPN is one of the best ways to model agent interaction protocols. However the notion of an agent executing an action with Petri Net is not explicit in the notation [18]. Different PNs can be assigned to each agent role, raising questions about how the entire protocol is inferred and the reachability and consistency of shared places. If a single Petri net is partitioned for each role, this leads to a complex diagram where a partition is required for each agent identified. Furthermore, alternative actions and states such as either agree or reject but not both, cannot be expressed in standard Petri nets.

Keeping in mind complex systems characterized by complex interaction, asynchronism and concurrency we propose to use for the purposes of their description a special language PRALU [6, 7] having its background in the Petri net theory (expanded nets of free choice – EFC–nets investigated by Hack [19]) but possessing special means for keeping track of the current states of the conversation, receiving messages and initiating responses. The formal language PRALU combines properties of "cause-effect" models with Petri nets. It is intended for a wide application in engineering practice and is well suited for representation of the interactions involved in concurrent system, synchronization among them and then it is simple enough for understanding. The language PRALU supports hierarchical description of the algorithms that is especially important in the case of complex systems. At last, powerful software has been developed that provides correctness verifying, simulation, hardware and software PRALU-description's implementation [7, 8]. The review of obtained results it can be found in [8].

PRALU Language

Any algorithm in PRALU consists of sequences of operations to be executed in some pre-determined order. There exist two basic kinds of operations used in PRALU: acting operations "$\rightarrow A$" and waiting operations "$- p$". Action operation changes the state of the object under control, whereas a waiting operation is passive waiting for some event without affecting anything. In simple case A and p are conjunctive terms, so acting and waiting operations can be interpreted as waiting for event $p = 1$ and producing event $A = 1$. But A can be understood

too as a formulae defining operations to be performed and p as a predicate defining condition to be verified [20]. For example, acting and waiting operations could be specified by the expressions such as

"$-(a > b + c)$" and "$\rightarrow (a := b + c)$".

The sequences consisting of action and waiting operations are considered to be linear algorithms. For instance, the following expression means: wait for p and execute A, execute B, then wait for q and execute C:

$$-p \rightarrow A \rightarrow B -q \rightarrow C$$

In general, a logical control algorithm can be presented as an unordered set of chains α_j in the form

$$\mu_j : - p_j L_j \rightarrow \nu_j ,$$

where L_j is a linear algorithm, μ_j and ν_j denote the initial and the terminal chain labels represented by some subsets of integers from the set $M = \{1, 2, ..., m\}$: $\mu_j, \nu_j \subset M$ and the expression "$\rightarrow \nu_i$" presents the transition operation: to the chains with labels from ν_j.

Chains can be fulfilled both serially and in parallel. The order in which they should be fulfilled is determined by the variable starting set $N_t \subseteq M$ (its initial value $N_0 = \{1\}$ as a rule): a chain $\alpha_j = \mu_j : - p_j L_j \rightarrow \nu_j$ (that was passive) is activated if $\mu_j \subseteq N_t$ and $p_j = 1$. After executing the operations of the algorithm L_j, N_t gets a new value $N_{t+1} = (N_t \setminus \mu_j) \cup \nu_j$. The algorithm can finish when some terminal value of N is reached (one-element as a rule), at which time all chains became passive. But the algorithms can also be cyclic; they are widely used when describing production processes.

When the conditions $\mu_j \subseteq N_t$ and $p_j = 1$ are satisfied for several chains simultaneously these chains will be fulfilled concurrently. On the contrary chains with the same initial labels are alternative (only one of them can be fulfilled at a time), they are united in a sentence with the same label as will be shown below.

Thus PRALU allows concurrent and alternative branching, as well as merging concurrent and converging alternative branches. These possibilities are illustrated with the following examples of simplified fragments [20]:

Concurrent branching	Merging concurrent branching	Alternative branching	Converging alternative branching
1: ...→2.3	2: ...→4	1: – a...→2	2: ...→4
2: ...	3: ...→5	– $\bar{a}$ ...→3	3: ...→4
3: ..	4.5: ...		4: ...

In PRALU there are two syntactic constraints on chains that restrict concurrent and alternative brunching. If some chains are united in the same sentence (they have equal initial labels) they should have orthogonal predicates in the waiting operations opening the chains:

$$(i \neq j) \,\&\, (\mu_i \cap \mu_j \neq \varnothing) \rightarrow (p_i \,\&\, p_j = 0).$$

The other constraint is similar to the corresponding condition specific for extended nets of free choice (Hack [19]):

$$(i \neq j) \,\&\, (\mu_i \cap \mu_j \neq \varnothing) \rightarrow (\mu_i = \mu_j).$$

PRALU language has some more useful properties that come in handy for description of complex interaction protocols.

1. PRALU algorithms can be expressed both in graphical and symbolic forms.

2. PRALU language permits hierarchical descriptions. The two terminal algorithms (having the only terminal label) may be used as blocks (invoked as complex acting operations) in hierarchical algorithms.

3. In PRALU there exist some additional interesting operations that can be useful for description of interaction protocols. Those are suppression operations and some arithmetic operations. The first ones provide response on special events that can take place outside or within control system. Suppression operation ("$\rightarrow *$", "$\rightarrow * \gamma$", "$\rightarrow ' \gamma$") interrupts the execution all concurrently executed chains of the algorithm or those ones mentioned in γ, (or do not mentioned in γ – in the case of "$\rightarrow ' \gamma$"). This operation breaks the normal algorithm flow. Among arithmetic

operations it ought to be mention timeout operations (waiting for *n* unit times "– *n*") and counting operation that counts event occurrences.

Specifying Protocols in PRALU

For an example of an interaction protocol, consider an English auction [5]. The auctioneer seeks to find the market price of a good by initially proposing a price below that of the supposed market value and then gradually raising the price. Each time the price is announced, the auctioneer waits to see if any buyers will signal their willingness to pay the proposed price. As soon as one buyer indicates that it will accept the price, the auctioneer issues a new call for bids with an incremented price. The auction continues until no buyers are prepared to pay the proposed price, at which point the auction ends. If the last price that was accepted by a buyer exceeds the auctioneer's (privately known) reservation price, the good is sold to that buyer for the agreed price. If the last accepted price is less than the reservation price, the good is not sold.

In the case of the auction there are participants of two types: the initiator of the conversation – Auctioneer, and others – Buyers. So, we have two kinds of interaction protocols – those of Auctioneer and of Buyers. The last participants are peer and should be described with identical interaction protocols modeled in our case by the operation "Buyer".

Interaction protocol as a whole can be represented in PRALU as three complex acting operations – blocks. PRALU-blocks are exchanging with values of logical (binary) variables, only such variables are mentioned in them. Each block has some sets of input and output variables that are enumerated in brackets following the block name (the other variables of a block are its internal). Initialization of a complex acting operation is depicted by the fragment such as "$\rightarrow$*Buyer". The operation Buyer exists in as many copies as the number of participants of the auction, so the copies of the operation differ in their indexes only.

The modeling of the process of auction begins with the execution of "Main_process" triggering event that initiates the interaction protocol execution. Here the processes Auctioneer and $Buyer_n$s are executed concurrently. For the sake of simplicity we limit the number of buyers to two (in principle it can be simply increased). The process Auctioneer starts with sending the first message (start_auction) that is waited by others participants to continue communication.

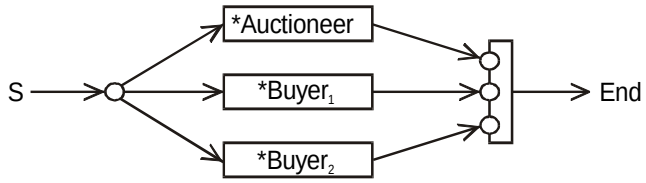
Below PRALU description of the auction interaction protocol is shown. Here we cite the only block $Buyer_n$, but for real application (intending to simulate the process of auction, for example) we should have as many proper copies as it has been used (in our case – two). Fig. 1 depicts graphical schemes of three mentioned PRALU-blocks: Main_process, Auctioneer and $Buyer_n$.

Main_process ()
1: $\rightarrow$2.3.4
2: $\rightarrow$*Auctioneer $\rightarrow$5
3: $\rightarrow$*$Buyer_1$ $\rightarrow$6
4: $\rightarrow$*$Buyer_2$ $\rightarrow$7
5.6.7: $\rightarrow$.

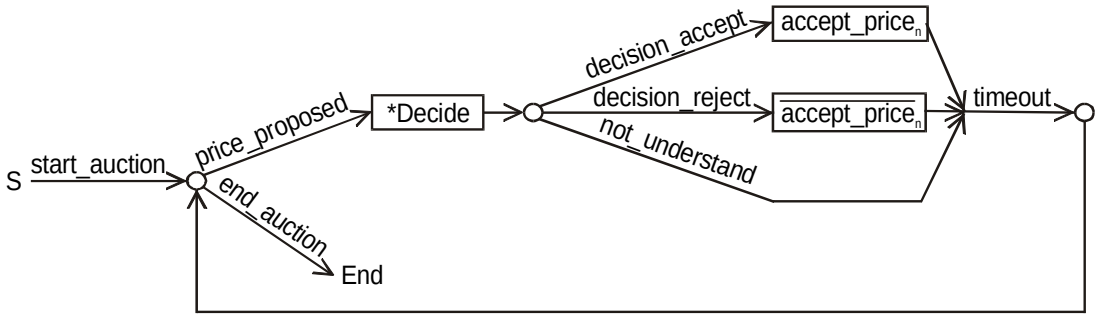
$Buyer_n$ (start_auction, price_proposed, end_auction / $accept\_price_n$, not_understand)
1: –start_auction $\rightarrow$2
2: –price_proposed $\rightarrow$*Decide(/decision_accept,decision_reject) $\rightarrow$3
–end_auction $\rightarrow$.
3: –decision_accept $\rightarrow$ $accept\_price_n$ $\rightarrow$4
–decision_reject $\rightarrow$'$accept\_price_n$ $\rightarrow$4
–not_understand $\rightarrow$4
4: –timeout $\rightarrow$2

Auctioneer ($accept\_price_1$, $accept\_price_2$, not_understand / start_auction, price_proposed, end_auction)
1: $\rightarrow$start_auction $\rightarrow$2
2: $\rightarrow$'$accept\_price_1$.'$accept\_price_1$.'not_understand $\rightarrow$*Price_propose(/price_proposed) $\rightarrow$3
3: –not_understand $\rightarrow$2
– $accept\_price_1$ $\rightarrow$4

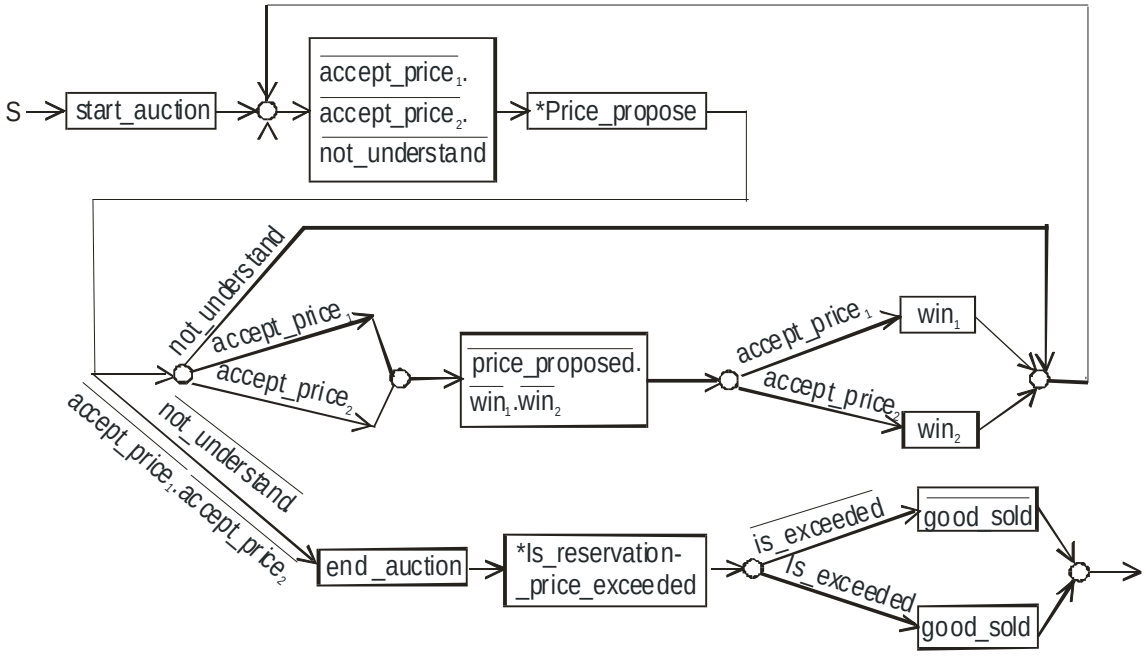
– accept_ price$_2$ →4
–′not_understand.′accept_ price$_1$.′accept_ price$_2$ →end_auction
→*Is_reservation_price_exceeded(/is_exceeded) →6
4: →′price_proposed.′win$_1$.′win$_2$ →5
5: – accept_ price$_1$ →win1 →2
– accept_ price$_2$ →win2 →2
6: –is_exceeded →good_sold →7
–′ is_exceeded →′good_sold →7
7: →.



Main_process ()



Buyer$_n$ (start_auction, price_proposed, end_auction / accept_price$_n$, not_understand)



Auctioneer (accept_price$_1$, accept_price$_2$, not_understand / start_auction, price_proposed, end_auction)

Fig. 1. English auction interaction protocol in PRALU

It is assumed that all unformalized operations are referred to as acting operations that set values of logical variables concerned with them. For example, Buyer's operation "Decide" decides for accepting or rejecting the announced price. Depending on adopted decision, it outputs true value of logical variable "decision_accept" or "decision_reject". In a similar, Auctioneers operation "Price_propose" proposes an initial price or increments the charged price outputting true value of logical variable "price_proposed"; the operation "Is_reservation_price_exceeded" verifies if the price accepted by a buyer exceeds the auctioneer's reservation price outputting true or false value of logical variable "is_exceeded".

The operation "–timeout" (where timeout is integer number) means waiting for timeout unit times before doing something followed it. The operation "→." is interpreted as the transition to an end of a process described by the block. When the processes of Auctioneer and all Buyers reach their end in the Main_process the transition to its end is executed.

Conclusion

This paper has addressed the need for formalized and more expressive logical and graphical methodologies for specifying (and then validating) interaction protocols in multi-agent systems. Towards this, it was proposed to use the formal language PRALU intended for the representation of complex interactions involved in concurrent system, being in need of synchronization among these interactions. It was demonstrated as well how PRALU algorithms could be used for the specification of multi-agent interaction protocols by the example of English auction. In favour of using the language PRALU is the existence of a great deal of methods and software developed for simulation and logical design of PRALU algorithms as well as for their hardware and software implementation.

References

1. B. Burmeister and K. Sundermeyer, "Cooperative problem-solving guided by intensions and perception", edited by E. Werner and Y. Demazean, Decentralized A.I. 3, Amsterdam, The Netherlands, North Holland, 1992.
2. V. Lesser, "Cooperative Multiagent Systems: A Personal View of the State of the Art", IEEE Trans. Knowledge and Data Engineering, vol. 11, no 1, pp. 133–142, 1999.
3. T. R. Gruber, "A Translation Approach to Portable Ontologies", Knowledge Acquisition, 1993, vol. 5, no 2, pp. 199-220, 1993
4. Finin, Y. Labrou and J. Mayfield, "KQML as an agent communication language", edited by J.M. Bradshaw, Software Agents, MIT Press, pp. 291–316, 1997.
4. ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group, July 1993.
5. Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org.
6. A.D. Zakrevskij, V.K. Vasiljonok, "The formal description of logical control algorithms for designing discrete systems", Electronnoje modelirovanie, vol. 6, no 4, pp. 79–84, 1984 (in Russian).
7. A.D. Zakrevskij, "Parallel algorithms for logical control", Minsk, Institute of Engineering Cybernetics of NAS of Belarus, 202 p., 1999 (in Russian).
8. L.D. Cheremisinova, "Realization of parallel algorithms for logical control", Minsk, Institute of Engineering Cybernetics of NAS of Belarus, 246 p., 2002 (in Russian).
9. M. Greaves, and J. Bradshaw J., "Specifying and Implementing Conversation Policies", Autonomous Agents '99 Workshop, Seattle, WA, May 1999.
10. G. Booch, J. Rumbaugh, and I. Jacobson, "*The Unified Modeling Language User Guide*", Addison Wesley, 1999.
11. B. Bauer, J.P. Müller, J. Odell, "Agent UML: A Formalism for Specifying Multiagent Interaction," *Agent-Oriented Software Engineering* , edited by P. Ciancarini and M. Wooldridge, Springer-Verlag, Berlin, pp. 91–103, 2001.
12. D. Harel and M. Politi, "Modeling reactive systems with statecharts", McGraw-Hill, 1998.
13. K. Jensen, "Colored Petri Nets – Basic Concepts, Analysis Methods and Practical Use", vol. 1: Basic Concepts, Springer-Verlag, Berlin, 1992.

14. Quan Bai, Minjie Zhang and Khin Than Win, “A Colored Petri Net Based Approach for Multi-agent Interactions”, 2nd International Conference on Autonomous Robots and Agents, Palmerston North, New Zealand, December 13-15, pp. 152–157, 2004
15. R. Cost, “Modelling Agent Conversations with Coloured Petri Nets”, Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Seattle,Washington, pp. 59–66, 1999.
16. D. Poutakidis, L. Padgham, and M. Winikoff, “Debugging Multi-Agent Systems Using Design Artefacts: The Case of Interaction Protocols”, Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi Agent Systems, Bologna, Italy, pp. 960–967, 2002.
17. J. Hutchison and M. Winikoff, “Flexibility and Robustness in Agent Interaction Protocols”, Proceedings of the 1st International Workshop on Challenges in Open Agent Systems, Bologna, Italy, 2002.
18. S. Paurobally, J. Cunningham, “Achieving Common Interaction Protocols in Open Agent Environments”, *AAMAS* ’02, Melbourne, Australia, 2002.
19. M. Hack, ”Analysis of production schemata by Petri nets”, Project MAC-94, Cambridge, 1972.
20. A. Zakrevskij, V. Sklyarov, “The Specification and Design of Parallel Logical Control Devices”, Proceedings of PDPTA'2000, June, Las Vegas, USA, pp. 1635–1641, 2000.

Authors' Information

Dmitry Cheremisinov, Liudmila Cheremisinova – The United Institute of Informatics Problems of National Academy of Sciences of Belarus, Surganov str., 6, Minsk, 220012, Belarus, Tel.: (10-375-17) 284-20-76, e-mail: cher@newman.bas-net.by, cld@newman.bas-net.by

WEBCOMPUTING SERVICE FRAMEWORK

Evgenija Popova

*Abstract: Presented is webComputing – a general framework of mathematically oriented services including remote access to hardware and software resources for mathematical computations, and web interface to dynamic interactive computations and visualization in a diversity of contexts: mathematical research and engineering, computer-aided mathematical/technical education and distance learning. webComputing builds on the innovative web*Mathematica *technology connecting technical computing system* Mathematica *to a web server and providing tools for building dynamic and interactive web-interface to* Mathematica*-based functionality. Discussed are the conception and some of the major components of webComputing service: Scientific Visualization, Domain-Specific Computations, Interactive Education, and Authoring of Interactive Pages.*

Keywords: web-access, mathematical user-interfaces, web computations, mathematical active learning.

ACM Classification Keywords: F.1.2 Modes of Computation: interactive computation, online computation; G.4 Mathematical Software: user interfaces; K.3.1 Computer Use in Education: distance learning.

Introduction

Internet and the World-wide Web make many kind of information and services easily accessible. However many Internet/Web technologies, so powerful in many areas, are not well suited to scientific computation; it is simply not their main focus. The importance of technical/mathematical communication on the Internet is underscored by the activities at the W3 consortium. Internet Accessible Mathematical Computations is part of these activities directed