Annales UMCS Informatica AI IX, 1 (2009)
111–122; DOI: 10.2478/v10065-009-0008-x

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Analysis of communication processes in the multi–agent systems

Wojciech Pieprzyca*

University of Computer Science and Management,
ul. Fredry 51, 43–346 Bielsko–Biała, Poland
PhD Studies in Computer Science, Faculty of Automatic Control,
Electronics and Computer Science Silesian University of Technology, Gliwice, Poland

Abstract

The article deals with the problems related to the possibility of information exchange in the multi–agent systems. Therefore it presents a model of coloured Petri net which was created in order to illustrate and to simulate the agents communication processes as well as exemplary diagrams which determine the sequences of information exchange between the agents with the use of the required performatives of KQML.

1. Introduction

A great part of the multi–agent systems effectiveness is based on the appropriate cooperation and coordination of the executed tasks. To make it possible you have to give access to the appropriate methods of the agents communication, e.g. define the language the agents will use to communicate and to exchange messages and knowledge. One of the possible languages is KQML (Knowledge Query Manipulation Language).

The KQML was designed by the group Knowledge Sharing Effort working under the auspices of the DARPA (Defense Advanced Research Projects Agency)

* *E-mail address:* wojtek@wsi.edu.pl

[**1**]. In the agent systems this language is used to exchange messages and knowledge between autonomous agents. Queries and orders emitted in KQML act on the basis of knowledge related to a given agent. In particular, it does not have to be a knowledge base in the traditional sense, but also so called virtual knowledge base, which is defined as an ordinary structure of an agent program or treated as information contained in the standard database. Queries can enable a check of the knowledge base content, its update etc.

In KQML, a message (an information exchange unit) is defined as a performative. This concept, which is derived from the act of speech theory, is defined in lingustics–related science. In general, performatives can be divided into:

- assertive messages (statements),
- directive messages (commands, questions, suggestions),
- declarative messages (information about the sender's abilities).

KQML has a predefined set of performatives the actions of which are determined and cannot be modified. However, it is possible to extend such a set, i.e. depending on the needs one can define one's own performatives, the action of those performatives will be determined within a specified agent system.

In KQML, a performative is a sequence of ASCII characters according to the grammar which was defined for this language and which is based on the Polish prefix notation. Every message, apart from its name, which identifies the performative type, has a set of parameters under the form of :name value e.g.: sender agent1 (parameter defining the name of the sender of the message). The order of parameters is not relevant. Table 1 presents the parameters of messages that occur most often in KQML.

Table 1. Basic parameters of performatives in KQML [**2**]

Parameter	**Meaning**
:sender	The sender of the message
:receiver	The receiver of the message
:reply-with	If this parameter does not occur or its value is nil, it means that the sender does not expect a reply to the message. Otherwise, the field contains an identifier to which the sender will refer while replying to the message (by writing this identifier in the field :in-reply-with).
:in-reply-to	The identifier defining which message the agent is replying to
:language	The language in which the message content is represented (:content)
:ontology	The name of the ontology which the message content (:content) refers to
:content	The message content
:force	Determines if the sender can modify the performative meaning in the future.

The message content itself is transparent for KQML and may be represented in various languages, using the format of ASCII characters or the binary code, it does not belong to the KQML standard. However, the most common languages for the content field :contents are the those that enable representation of the knowledge base content e.g. KIF (Knowledge Interchange Format), Prolog etc.

The KIF language is based on the first order logic. It was developed by the group KSE (Knowledge Sharing Effort) which works on languages representing knowledge with an emphasis on the problem of knowledge exchange between the heterogeneous information systems.

Among the crucial characteristics of the KIF language are [**3**]:

– implementation which does not depend on semantics,
– expressiveness, a possibility of translating and representing most of the knowledge representation systems, -
– the language is legible for people.

An example of KIF expressions:

- Temperature m1 = 83 Celsius
  (= (temperature m1) (scalar 83 Celsius))
- X is a bachelor if he is a man and he is not married
  (⇒ (and (man ?x) (not (married ?x))) (bachelor (?x))

2. A message in the KQML

A message in KQML is composed of a list of elements surrounded by a pair of brackets and separated with spaces. The first element of the message is the determination of the type of performative, all the following elements of the list are parameters.

```
(ask–one
  :sender A
  :receiver B
  :reply–with 101
  :language KIF
  :ontology book
  :content (price ISBN1-58053-605-0 ?x))
```

In the exemplary message above, ask-one is the performative. The sender of the message is the agent A, the message is sent to the receiver called B. Additionally, the name of the language (KIF) and the ontology (book) were determined, they express the message content (:content). The content field

contains a query formulated in the KIF language and related to the price of a book with the given ISBN number.

3. An agent system model

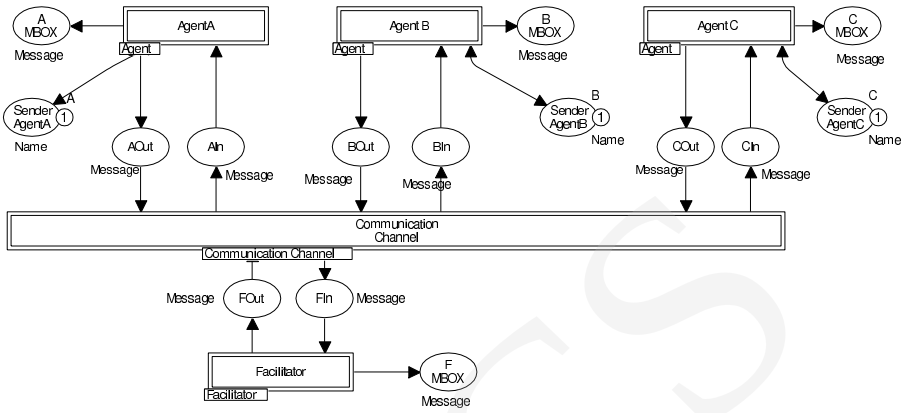

Fig. 1. Multi-agent system

The presented model was created with the use of the coloured Petri net. Its main elements are 3 hierarchical sites: *Agent* – represents the agent's action, *Communication Channel* – represents the way of sending messages between the agents and *Facilitator* – represents the way of executing the tasks by the facilitator agent.

In order to simplify things the system is considered to have three ordinary agents (A,B,C) and one facilitator agent (*Facilitator*). The sent messages are transmitted to the given agents' boxes (*A MBOX*, *B MBOX*, *etc.*).

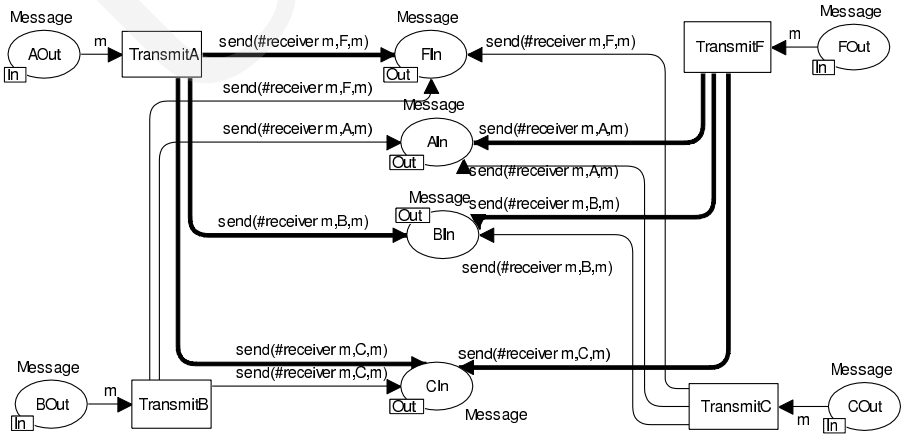

Fig. 2. Communication channel

In the communication channel (Fig. 2) the transport of messages between the agents takes place. The transitions *Transmit* cause the token (of the message) to be taken out of the agent–sender's place *Out* and to be added to the place *In* of the agent–receiver. The message is transmitted only if the function *send* determines that the agent is the receiver of the given message.

4. An agent model

Every ordinary agent is modelled as an instance of the subsite Agent (Fig. 3). During the simulation the user selects, out of nine performatives, the one to be created. Depending on the selected message type, it is created in one of the following subsites: *prepare msgToA* (performatives addressed to other ordinary agents), *prepare msgToF* (performatives handled by the facilitator agent) or *prepare standby* (the standby performative). A message thus created is transmitted to the place *Message Ready* from which it can be sent forward to the communication channel by launching the transition *Send Performative*.

An incoming message can be received by launching the transition *Receive Performative* which causes the token (of the message) to be added to the agent's box (*MBOX*). Depending on the type of the sent performative, attached to the message, the following service takes place on the subsite *answerAsk, answerRecommend, answerStreamall* or *answerStandby*. In the preliminary state only the transition *Choose performative* is active.

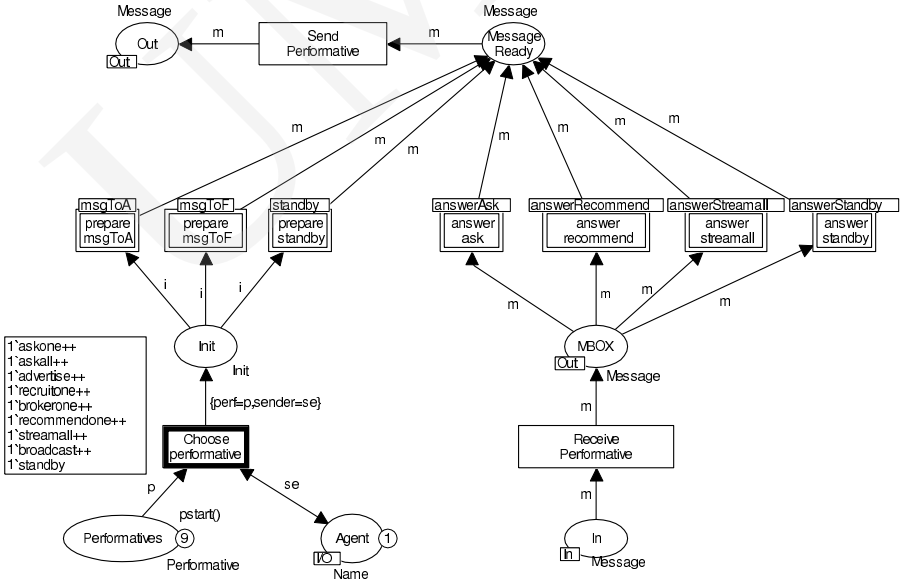

Fig. 3. Agent model

5. Basic performatives

Basic performatives enable the creation of simple queries addressed to the agent's knowledge base [**2**].

a) **ask–one**– the sender asks the receiver to answer the question contained in the field :content. The answer is created on the basis of the receiver's knowledge base.
b) **tell**– indicates that the content of the field :content is written in the agent-sender's knowledge base.
c) **ask–all**– its action is similar to that of ask-one, but the answer contains the collection of all the theorems from the receiver's knowledge base which correspond to the query sent by the sender in the field :content.
d) **stream–all**– its action is comparable to that of ask–all, but in this case the agent does not send the entire collection of corresponding theorems, it sends a series of subsequent performatives instead. The last message sent within the series is the performative eos (end of stream).

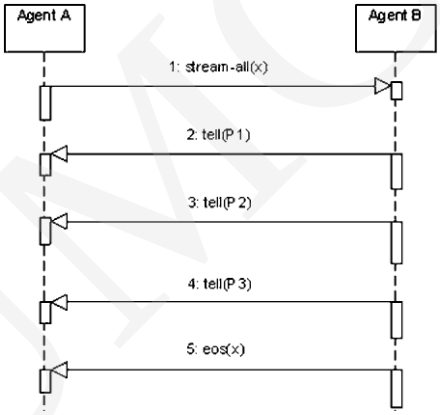

Fig. 4. A sequence of message exchanges with the use of the stream-all performative

In the model of Petri net the subsite *prepare msgToA* (Fig. 5) is responsible for creating the above messages. On this subsite the transition *create message* is launched only for the above mentionned performatives by determining an appropriate condition of exciting the transition (a so called surveillance, in the figure the condition of surveillance is surrounded by square brackets).

A message is created with an additional use of the place *ID*, which is destined to generate unique id values for each of the sent messages. The operation is based on the principle of autoincrementation i.e. each following message receives an id number which is greater by one than the previous number. This place is shared by all the instances of the agents (a so called *fusion place*), which

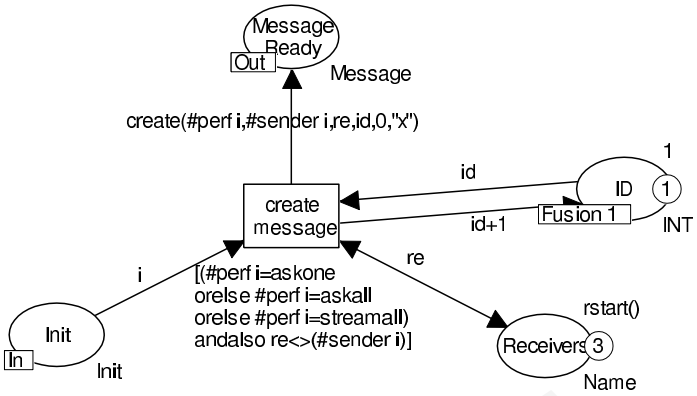

Fig. 5. Message creating (ask-one, ask-all, stream-all)

means that the number of the last generated id is global and available in all the instances of the agents. From the place *Receivers* there comes the information about the message receiver. The launch of the transition *create message* results in exciting the function *create*, which creates the final message on the basis of the received parameters.

The answers to the performatives answer–one, answer–all and stream–all are generated on the subsites *answerAsk* and *answerStreamall*. In the case of the performative answer–all all the answers that correspond to the query are put together and sent as one message. It is not the case of the stream–all performative for which every answer is sent as a separate message.

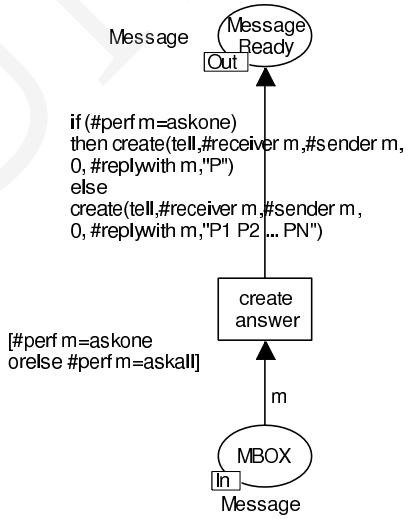

Fig. 6. Answer creating (the answer–one and answer–all performatives)

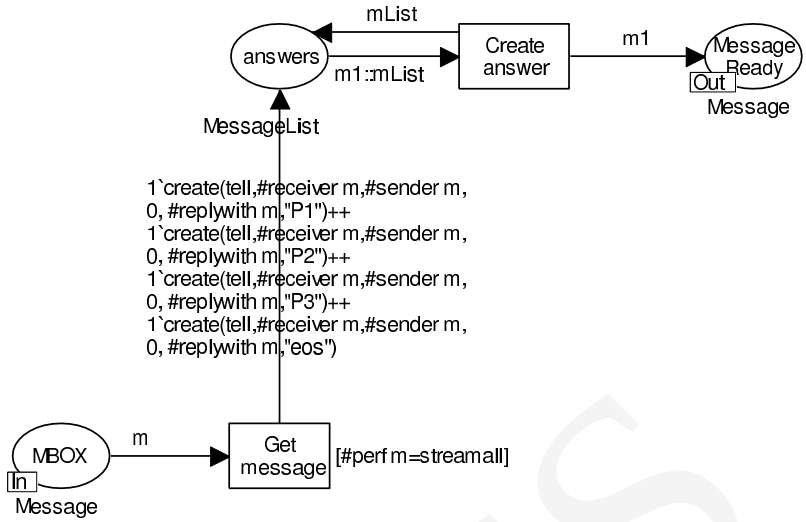

Fig. 7. Answer creating (the stream–all performative)

5.1. Conditional stream generating answers to the queries.

In certain cases the sender of the query wants to receive the answer as a stream (like in the case of the stream-all performative) and, moreover, have an influence on the moment of receiving following answers. It is possible thanks to the performatives described below [**2**]:

a) **standby**– the sender applies for the preparation of an answer to the performative contained in the field :content. The answer is not generated immediately, but only after the receiver has announced its readiness to give an answer (using the performative ready)
b) **ready**– the sender considers itself to be ready to give an answer to the performative identified by the content of the field :in–reply–to
c) **next**– the sender indicates that it expects a following answer which has been prepared (according to the performative ready which had been received earlier)
d) **rest**– the sender indicates that it expects the rest of the answers under the form of a series of performatives contained in one message
e) **discard**– the sender discards the other answers which have been generated in reply to the previous standby order.

In the diagram of Fig. 8 the agent B, having prepared a collection of answers, does not send them immediately, they are sent one by one as a reaction to sebsequent performatives next. The performative eos communicates the end of sending a series of messages. If agent A wants, at a given moment, to receive all the other answers of the series as one message, it can achieve it sending the

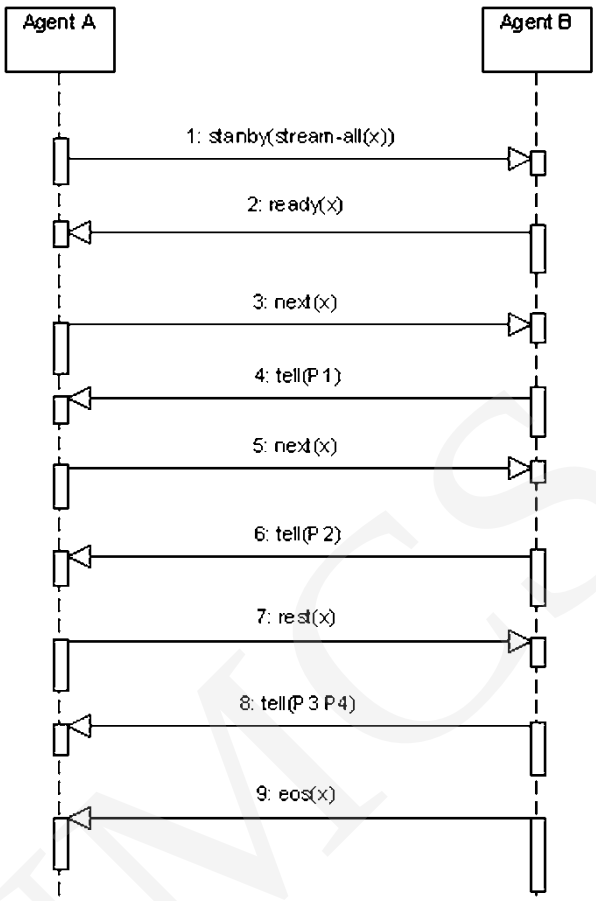

Fig. 8. A sequence of message exchanges with the use of the standby performative

rest performative. Agent A can send the discard performative in order to stop receiving the following answers of the series also.

6. Performatives using the facilitator agent

The facilitator agent (*Facilitator*) is a special type of agent which has the information about all the agents that exist in a given system, about their addresses and abilities and that is why it is vastly used in the processes of searching agents which have determined abilities and resources and offer access to specified services. The following performatives are related to the facilitator agent in KQML [**2**]:

a) **broadcast**– the sender asks the receiver to send the emitted performative to all the agents it is connected to. In the case of a modelled system this

performative is received by the facilitator agent, which transmits the message to all the agents of the system.

b) **advertise**– the sender announces its readiness to give answers to a determined type of performatives (determined in the field :content).

c) **broker–one**– the sender asks the receiver to transmit, with the purpose of its execution, the attached performative to one agent which has required abilities and resources (the agent with such characteristics will have sent the advertise performative with appropriate parameters). In Fig. 9 the facilitator agent transmits the execution of the ask–one(x) performative to agent A because the latter one declared, using the advertise performative, the ability to handle this kind of queries. Afterwards the answer is sent to the facilitator agent which transmits it to the sender of the query, i.e. agent B.

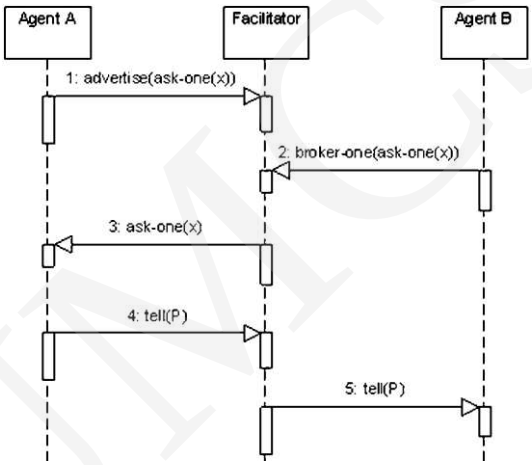

Fig. 9. A sequence of message exchanges with the use of the broker-one performative

d) **recommend–one**– the sender asks the receiver to give the name of one agent which is able to execute the attached performative.

e) **recruit–one**– acts in the same way as broker–one, the only difference being the fact that the answer is sent directly from the agent which has the required abilities to the sender of the performative recruit–one.

The model of facilitator agent is on the subsite *Facilitator* and contains the mechanisms that offer the possibility to give answers and process the above described performatives. If the facilitator agent does not find an agent with the required ability, the sorry performative will be generated as an answer. The information about the agents' abilities are stored in the place *Advertised*.

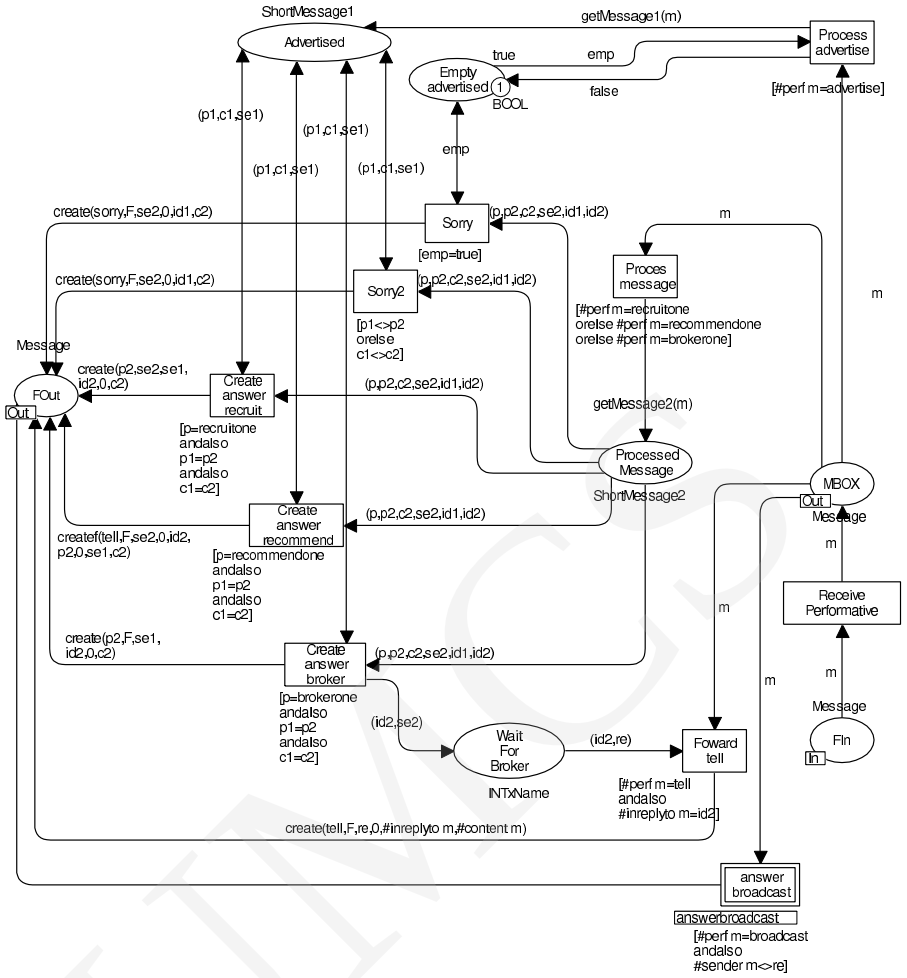

Fig. 10. Facilitator agent model

7. Conclusions

The idea of languages based on performatives which determine their functionality is used to exchange information and knowledge between intelligent program units, e.g. the agents. On the basis of KQML the international standardization organism FIPA (Foundation for Inteligent Physical Agents) developed the language FIPA ACL, which nowadays is considered to be a standard of the communication between agents in the multi–agent systems.

The created model offers a possibility to design a multi-agent system with the use of the performatives. A series of simulations of sending messages with the use of all the performatives was conducted. A space of available system states also was determined. The above described analysis did not lead to the

discovery of any inactive transitions (events) or undesired system states. The analys's of the system states proved that all the states resulting from the system functionality are possible to achieve. The above described model can be implemented in any high–level programming language. Until now KQML has been successfully applied in the environments for designing and programming agents, such as Saci, JATLite, Jackal and others. An important element of the presented system, apart from the language itself, is a special kind of facilitator agent (*Facilitator*), which, thanks to its knowledge about the possibilities and the actions of the agents, can coordinate the cooperation of agents within the whole agent system.

The full version of the presented model of the multi–agent system using the messages of KQML is available on the website http://wojtek.wsi.edu.pl/petri1. Its initiation requires the tool CPN Tools, version 2.2.0 or higher.

References

[1] Bradshaw J., *Software Agents*, MIT Press, Cambridge 1995.

[2] DARPA Knowledge Sharing Initiative External Interfaces Working Group: DRAFT Specification of the KQML Agent–Communication Language, 1995.

[3] Finin T., Labrou Y., *Agent Communication Language*, First International Symposium on Agent Systems and Applications and the Third International Symposium on Mobile Agents, 1999.