Formal Social Norms and Their Enforcement in Computational MAS by Automated Reasoning

Roman Neruda, Ondřej Kazík

***Abstract*—Role-based frameworks enrich multi-agent system models with new organizational concepts: roles and groups. We present a formalization of a role-based approach in description logics serving as the common organizational model both for the development process and the management of the system. A central authority of role ontology agent is proposed in this paper. This agent performs management of the system: keeping track of the actual state of the system, querying the model and checking its social norms. Agents in the system can thus act in a sociable way especially in complicated configurations. Algorithms controlling agents in a computational intelligence modeling scenario are shown.**

***Index Terms*—role model; description logic; integrity constraints; computational intelligence.**

I. Introduction

AGENT is a computer system situated in some environment that is capable of autonomous action in this environment in order to meet its design objectives [1]. Its important features are adaptivity to changes in the environment and collaboration with other agents. Interacting agents join in more complex societies, *multi-agent systems* (MAS). These groups of agents gain several advantages, as are the applications in distributed systems, delegacy of subproblems on other agents, and flexibility of the software system engineering. The effort to reuse MAS patterns brings the need of separation of the interaction logic from the inner algorithmic logic of an agent. There are several approaches providing such separation and modeling MAS from the organizational perspective, such as the *tuple-spaces*, *group computation*, *activity theory* or *roles* [2].

Generally speaking, a role is an abstract representation of stereotypical behavior common to different classes of agents. Moreover, it serves as an interface, through which agents perceive their execution environment and affect this environment. Such a representation contains a set of patterns of interactions, *capabilities*, and knowledge which an associated agent may utilize to achieve its goals. On the other hand, the role defines *constraints*, which a requesting agent has to satisfy to obtain the role, as well as responsibilities for which the agent playing this role holds accountable. The role also serves as a mean of definition of protocols, common interactions between agents. An agent may handle more roles, and a role can be embodied by different classes of agents. Moreover, agents can change their roles dynamically.

The role-based solutions may be independent of a particular situation in a system. This allows to design an overall organization of multi-agent systems, represented by roles and their interactions, separately from the algorithmic issues of agents and to reuse the solutions from different application contexts. The coordination of agents is based on local conditions, namely the positions of an agent playing some role, thus even a large MAS can be built out of simple organizational structures in a modular way.

The above mentioned properties are crucial for agent use in the computational intelligence models for data mining where we typically have a collection of available methods that cooperate in order to build a hybrid model based on data. Hybrid models including combinations of computational intelligence methods, such as neural networks, genetic algorithms, and fuzzy logic controllers, can be seen as complex systems with large number of components and computational methods, and with potentially unpredictable interactions between these parts. Typical set up of a data mining task employing a computational method contains the agent encapsulating such a method, such as multilayer perceptron neural network, or an RBF (radial basis function) network which is used throughout our example; the data source; and some managing entity responsible for proper configuration and execution of the task. These approaches have demonstrated better performance over individual methods in many data modeling tasks [3]. The modes of cooperation are partially expressed by ontological constraints, while some aspects are dynamical and data dependent.

There are not many software packages that provide a large collection of individual computational methods, as well as the possibility to connect them into hybrid schemes in various ways. Multi-agent systems seem to be a suitable solution to manage the complexity and dynamics of hybrid systems.

The next section discusses the work related to the role-based methodologies, their formalization and implementation support. In Section III the computational intelligence and its role-based model are introduced. In Section IV this model is formalized in description logic with respect to both open world and closed world reasoning. The role ontology agent dealing with this model is introduced in Section V. In Section VI typical agents exploiting services of the ontology agent are shown.

II. Related Work

There are various methodologies exploiting concept of role for development of MAS. These methodologies vary in their support of different phases of the development. For example, the Gaia methodology [4] fully exploits roles only in the analysis phase and leaves them during the design phase of development. The BRAIN framework [5], one of the role-based approaches, describes roles by means of XML-files and offers also the implementation support in JAVA language. The ALAADIN framework [6] is a organization-

R. Neruda is with The Institute of Computer Science, Academy of Sciences of the Czech Republic, 182 07 Prague 8, Czech Republic, e-mail: roman@cs.cas.cz. This work has been supported by the Czech Science Foundation under the project no. . O. Kazík is with The Faculty of Mathematics and Physics, Charles University, 118 00 Prague 1, Czech Republic, e-mail: kazik.ondrej@gmail.com

centered generic meta-model of multi-agent systems. It defines a general conceptual structure which is utilized in the MAS development. The framework describes MAS from an organizational perspective, instead of using terms of agents' mental states (agent-centered). This model (also called AGR) focuses on three basic concepts: agent, group and role.

In [7], the OWL-based representation of policies and norms in MAS, called OWL-POLAR was proposed. The OWL-POLAR also uses reasoning mechanisms to make policy-governed decisions and policy analysis. This formalization is near to the role-based frameworks but, in order to represent policy it has to express it in other formalism, namely SPARQL queries.

The matchmaking, which is one of the functions provided by the ontology agent of our system described in Section V, is also well established discipline in the field of web services. Various middle agents are presented in [8].

Data-mining algorithms and their interfaces are described in a domain ontology KDDONTO [9], based on description logics. This ontology is used for automatic composition of algorithms forming valid data-mining processes

We focused our analysis on the field of hybrid intelligence and computational MAS. In these research areas is laid emphasis on collaboration of different heteronomous computational methods. The multi-agent systems allow distributing not only the computation but also the data sources sensing in real time. This would enable ubiquitous and pervasive computing scenarios. The distribution of sensors is extensively studied in wireless sensor network technology, e.g. [10].

In our approach, a computational MAS contains one or more computational agents, i.e. a highly encapsulated objects embodying a particular computational intelligence method, and collaborating with other autonomous agents to fulfill its goals. Several models of development of hybrid intelligent systems by means of MAS have been proposed, e.g. [11], [12], and [13].

III. Role Model of Computational MAS Scenario

In order to verify the abilities of role-based models we will present an example of analysis of a computational MAS scenario. We are exploiting the conceptual framework of the AGR model [6]. Its organization-centered perspective allowing modular and variable construction of MAS is well suited especially to more complicated configurations of computational agents.

As an example we take the computational MAS from [14]. The system consists of a Task Manager agent, Data Source agent, two computational agents (RBF neural network and Evolutionary algorithm agent) and supplementary agents. In the case of RBF network, there are unsupervised (vector quantization) and supervised (gradient, matrix inverse) learning agents. The evolutionary algorithm agent needs Fitness, Chromosome and Tuner agents. We want to decompose this scenario by means of AGR concepts, i.e. agents, groups and roles.

Such a computational MAS is represented by a role *organizational structure*, depicted in Figure 1, consists of the following group structures:

- *Computational Group Structure*. It contains three roles: a Task Manager, Computational Agent implementing a computational method and Data Source which provides it with training and testing data.
- *Simple Learning Group Structure* consisting of two roles: a Teacher and Learned Computational Agent. This structure can be instantiated by groups for different Teacher (e.g. Vector Quantization, Gradient and matrix inverse).
- *Evolutionary Algorithm Group Structure* contains an Evolutionary Algorithm agent, Evolved Computational Agent, Chromosome which translates representation of an individual into model's parameters, and Tuner with probabilities of the algorithm.

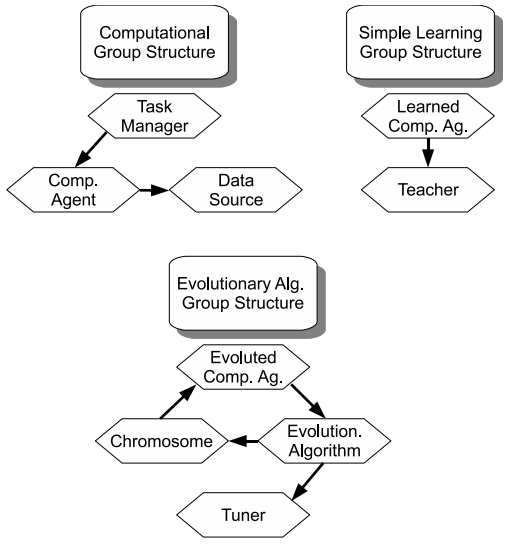

Fig. 1. Organizational structure diagram of a computational MAS.

Every concrete organization is built with respect to the rules of the organizational structures. Aims of the agents are fulfilled by assuming of roles or establishing of groups and interactions. The agents can play different roles in different groups and even complicated MAS can be built from these structures.

We can see, that the role model allows to simplify the construction of more complicated computational multi-agent systems by its decomposition to the simple group structures and roles, to which the agents assigns. Moreover, the position of an agent in MAS in every moment of the run-time is defined by their roles without need to take into account their internal architecture or concrete method it implements. It also reduces a space of possible responding agent when interactions are established.

IV. Description Logic Model

The family of *Description Logics* (DL [15]), fragment of first-order logic, is nowadays de facto standard for ontology description language for formal reasoning. In DL a knowledge base is divided into T-Box (terminological box), which contains expressions describing concept hierarchies, and A-Box (assertional box) containing ground sentences.

In [14], an ontological description of computational MAS has been proposed. The approach combines description logic and Horn rules in a single model providing both static and dynamic aspects of the system. This extension of the DL

model by other formalism is forced by the low expressive power of the OWL, the language based on the DLs, and its standard semantics. First of all, it does not use any variables, thus it is difficult to express more complicated conditions between entities in the model. Furthermore, semantics of OWL is designed for scenarios where the complete information cannot be assumed, thus it adopts the *Open World Assumption* (OWA). According to OWA, a statement cannot be inferred to be false only on the basis of a failure to prove it. If there is assumed complete knowledge, the T-Box axioms cannot be used as *Integrity Constraints* (IC) which would test validity of the knowledge base. In order to check integrity constraints the *Closed World Assumption* (CWA) is necessary. There are several approaches simulating CWA by different formalisms, e.g. rules or queries [16].

We continue in the effort to describe the computational MAS in description logic model. Our model would incorporate the concepts of group and role elaborated in the previous Section. We want to preserve the simplicity of the OWL models and also to express role ICs in the same language. In [16] authors presented a IC validation solution reducing IC validation problem to SPARQL query [17] answering. Moreover, they introduced prototype IC validator extending Pellet [18], the OWL reasoner. Thus we divided the T-Box of proposed model into two parts. The first part contains axioms describing mainly concepts' hierarchy and the necessary relations between their instances. This schema is interpreted in OWA and defines the facts the reasoner will infer from the given A-Box. In the second part, there are constraints which define the integrity conditions of the system related mainly to the capabilities of agents. These are interpreted on CWA. The time-dependent information, the actual state of the system is in A-Box of the ontology.

As we have already mentioned, a role is defined as a set of capabilities, i.e. actions (interactions) an agent assuming this role can use, and set of responsibilities or events the agent should handle. A group is then described by a set of the roles, the group contains. A hierarchy of concepts should respect this. The designed T-Box contains the following superior concepts:

- $Responder$ is a responsibility of a role. It stands for a message type the agent handles.
- $Initiator$ represents an action from a capability set and it is closely related to a particular $Responder$. The functional role $isInitiatorOf$ relates to the agent which the action uses. The role $sendsTo$ contains the agents to which the action is connected.
- $RequestInit$ is a subclass of the previous concept which defines only those initiators that sends messages to one agent (unlike e.g. the contract net protocol). This concept adds the following IC:

$$RequestInit \sqsubseteq_C \leqslant sendsTo.1$$

- $Agent$ is a superclass of all roles. The role assignment is achieved simply by a concept assertion about the agent. The inverse functional roles $hasInitiator$ (inverse of $isInitiatorOf$) and $hasResponder$ couple an agent with particular actions and responsibilities. While the $hasResponder$ relation is a fixed property, the $hasInitiator$ occurs only when a corresponding connection is established. Finally, the functional role $isMemberOf$ indicates belonging to a group.
- $Group$ represents a group in a MAS. It has only one role, an inverse of the $isMemberOf$ role, called $hasAgent$.

The *computational group structure* contains three agents with assigned roles of a task manager, computational agent and data source. Between these roles two connections can be established. First, the task manager sends a control messages to the computational agent in order to solve a problem. It contains necessary parameters (data file name, learning options) and action the computational agent should perform, as is training and testing. The second connection is between the computational agent and data source, which provides data from a specified file. The sending of control messages between the task manager ($TaskManager$), which is initiating this connection, and computational agent ($CompAgent$) is modeled by two concepts, an initiator ($ControlMsgInit$) and responder ($ControlMsgResp$). The initiator of this connection is an instance of $ControlMsgInit$ which is a subclass of the $RequestInit$ class. It sends messages only to an agent with a running responder handling these messages, and it is coupled with a Task Manager role as a capability. The schema file of the ontology contains axioms of the initiator and responder concept hierarchy, and a definition of the responder individual:

$$ControlMsgInit \quad \sqsubseteq_O \quad RequestInit$$

The following integrity constraints for this concept check the roles of initiating and responding agents:

$$ControlMsgInit \sqsubseteq_C \forall sendsTo.\exists hasResponder.$$
$$.ControlMsgResp \sqcap \forall isInitiatorOf.TaskManager$$

The control message responder is a simple descendant of the $Responder$ concept and this class contains the instance $ControlMsg$. The schema axioms follow:

$$ControlMsgResp \sqsubseteq_O Responder$$
$$ControlMsgResp(ControlMsg)$$

The data connection between the computational agent and the source of data ($DataSource$) is divided in two classes: an initiator $DataMsgInit$ and responder $DataMsgResp$. The following axioms for these concepts are similar to those for the control connection:

$$DataMsgInit \sqsubseteq_O RequestInit$$
$$DataMsgInit \sqsubseteq_C \forall sendsTo.\exists hasResponder.$$
$$.DataMsgResp \sqcap \forall isInitiatorOf.CompAgent$$
$$DataMsgResp \sqsubseteq_O Responder$$
$$DataMsgResp(DataMsg)$$

Role definitions are descendants of the $Agent$ concept and have to contain their responsibilities, i.e. responders they contain (capabilities are defined on the initiator side). The computational agent ($CompAgent$) has as a responsibility to respond on the control connections. These are axioms inserted in the schema set:

$$CompAgent \quad \sqsubseteq_O \quad Agent$$
$$\sqcap \quad \ni hasResponder.ControlMsg$$

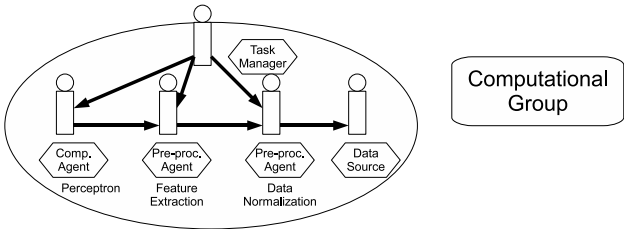

Fig. 2. Example of computational group solving data-mining task which employs two preprocessing agents in order to prepare data for the neural-network classification.

The data source ($DataSource$) handles requests for data and the task manager ($TaskManager$) role only sends messages in a group:

$$\begin{aligned} DataSource &\sqsubseteq_O Agent \\ &\sqcap \ni hasResponder.DataMsg \\ TaskManager &\sqsubseteq_O Agent \end{aligned}$$

Finally, the computational group ($CompGroup$) contains only the agents which have asserted that they have the computational agent, task manager or data source role. The subclass-axiom is important for open world reasoning:

$$CompGroup \sqsubseteq_O Group$$

On the other hand, entrance of the agent with a wrong role has to be checked by the following closed world constraint:

$$\begin{aligned} &CompGroup \sqsubseteq_C \forall hasAgent. \\ &\quad .(CompAgent \sqcup TaskManager \sqcup DataSource) \end{aligned}$$

In real data-mining tasks, separate phase of pre-processing before main computation is necessary [19], e.g. feature extraction, missing values and outliers filtering, or resampling etc. The *pre-processing agent*, i.e. encapsulation of a preprocessing method, obtains data from a data source and provides pre-processed data to other agents. The options of the pre-processing method and source-file have to be set by a task manager who controls the computation.

Therefore the interactions defined before can be utilized. The pre-processing agent gains properties of both the data source (it provides data) and computational agent (it receives data from another source and waits for control messages). Thus the role of $PreprocessingAgent$ is defined as an intersection of $DataSource$ and $CompAgent$:

$$PreprocessingAgent \sqsubseteq_O CompAgent \sqcap DataSource$$

The pre-processing agent with this role is also able to enter any computational group according to this definition. It also includes the possibility of creation of agents chain, where on the one end is an agent providing original data table and on the other is a data mining computational method. Diagram of such a configuration with two preprocessing agent is at Figure 2.

The *simple learning group structure* is defined in a similar way with following schema and integrity rules:

$$\begin{aligned} &LearningMsgInit \sqsubseteq_O RequestInit \\ &LearningMsgInit \sqsubseteq_C \forall sendsTo.\exists hasResponder. \\ &\quad .LearningMsgResp \sqcap \forall isInitiatorOf.LearnedCA \\ &LearningMsgResp \sqsubseteq_O Responder \\ &LearningMsgResp(LearningMsg) \\ &LearnedCA \sqsubseteq_O Agent \\ &Teacher \sqsubseteq_O Agent \\ &\quad \sqcap \ni hasResponder.LearningMsg \\ &SimpleLearningGroup \sqsubseteq_O Group \\ &SimpleLearningGroup \sqsubseteq_C \forall hasAgent. \\ &\quad .(Teacher \sqcup LearnedCA) \end{aligned}$$

The *evolutionary algorithm group structure* is example of more complicated organization and it contains an evolutionary algorithm, evolved computational agent, tuner with parameters of the algorithm, and chromosome, i.e. representation of individuals. The interactions between these roles describe the necessary flow of information to evolutionary learn a model of the computational agent, i.e. control messages of the algorithm, messages with parameters of the algorithm, requests for fitness computation of single individual, and computation of error rate of CA with chromosome transformed to its model. The structure is defined as follows:

$$\begin{aligned} &EAControlMsgInit \sqsubseteq_O RequestInit \\ &EAControlMsgInit \sqsubseteq_C \forall sendsTo.\exists hasResponder. \\ &\quad .EAControlMsgResp \sqcap \forall isInitiatorOf.EvolvedCA \\ &EAParamsMsgInit \sqsubseteq_O RequestInit \\ &EAParamsMsgInit \sqsubseteq_C \forall sendsTo.\exists hasResponder. \\ &\quad .EAParamsMsgResp \sqcap \forall isInitiatorOf.EvoAlgorithm \\ &FitnessMsgInit \sqsubseteq_O RequestInit \\ &FitnessMsgInit \sqsubseteq_C \forall sendsTo.\exists hasResponder. \\ &\quad .FitnessMsgResp \sqcap \forall isInitiatorOf.EvoAlgorithm \\ &ComputeModelMsgInit \sqsubseteq_O RequestInit \\ &ComputeModelMsgInit \sqsubseteq_C \forall sendsTo.\exists hasResponder. \\ &\quad .ComputeMsgResp \sqcap \forall isInitiatorOf.Chromosome \\ &EAControlMsgResp \sqsubseteq_O Responder \\ &EAControlMsgResp(EAControlMsg) \\ &EAParamsMsgResp \sqsubseteq_O Responder \\ &EAParamsMsgResp(EAParamsMsg) \\ &FitnessMsgResp \sqsubseteq_O Responder \\ &FitnessMsgResp(FitnessMsg) \\ &ComputeModelMsgResp \sqsubseteq_O Responder \\ &ComputeModelMsgResp(ComputeModelMsg) \\ &EvolvedCA \sqsubseteq_O Agent \\ &\quad \sqcap \ni hasResponder.ComputeModelMsg \\ &EvoAlgorithm \sqsubseteq_O Agent \\ &\quad \sqcap \ni hasResponder.EAControlMsg \\ &Chromosome \sqsubseteq_O Agent \\ &\quad \sqcap \ni hasResponder.FitnessMsg \\ &Tuner \sqsubseteq_O Agent \end{aligned}$$

$$\sqcap \ni hasResponder.EAParamsMsg$$
$$EvoAlgorithmGroup \sqsubseteq_O Group$$
$$EvoAlgorithmGroup \sqsubseteq_C \forall hasAgent.$$
$$.(EvolvedCA$$
$$\sqcup EvoAlgorithm$$
$$\sqcup Chromosome$$
$$\sqcup Tuner)$$

Fig. 3. Hierarchy of DL concepts in the computational MAS model.

We define concepts in this hierarchy in order to describe our computational MAS. The system consists of groups with three types: computational group, simple learning group and evolutionary algorithm group. Behavior of the agents is formally described by means of allowed roles and protocols of their interactions in these groups. The full formal model is presented in [20]. The main concepts of this ontology are depicted in Figure 3.

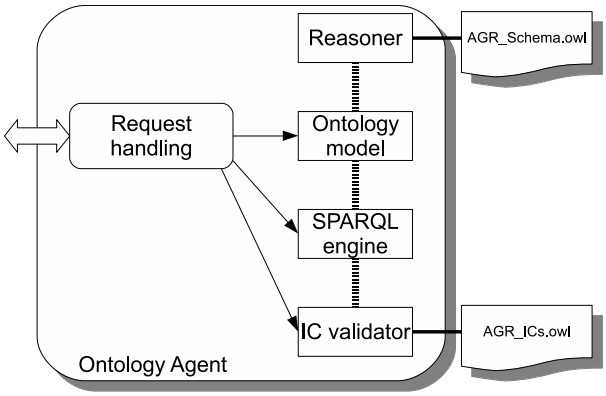

Fig. 4. Architecture of the role ontology agent.

V. Implementation

To coordinate the run-time role organization of MAS built according to the schemes and constraints of T-Box, it is necessary to have a central authority, separate agent in which the DL model has to be represented. Other agents will change the state of the model and query it by interaction with this agent.

The model is implemented as an *role ontology agent* (ROA) in JADE, Java-based framework for MAS [21]. The goals of the ROA are:

- Keeping track of the actual state of MAS. Agents present in the MAS are registering themselves in ROA; stating changes of their roles; creating and destroying groups and their membership in them; and establishing communication channels.
- Verification of correctness of MAS. ROA controls all changes of the system and does not allow activities which would violate the integrity constraints.
- Matchmaking of agents and groups. Exploiting the concept hierarchy it is possible to search groups of certain types or agents that have a certain role, are members of certain group, or can handle certain types of messages.

The ontology agent (shown in Figure 4) consists of the *request handling* module which is responsible for processing of incoming requests and replying. It employs the ontology functions provided by the Pellet OWL-DL reasoner [18] and its extensions. The *ontology model* contains assertional box of the ontology and describes an actual state of the system. The open-world *reasoner* infers new facts from axioms in the OWL schema file and content of the A-Box. The integrity constraints saved in separate OWL file are converted into SPARQL queries and run by the *SPARQL engine* on the ontology model. The SPARQL engine is also used to answer matchmaking queries.

The communication ontology for contents of ROA messages has been created. This ontology consists of three types of concepts.

The first group contains *actions* changing state of the ontology. These actions result in changing of assertions in the A-Box of the model and are validated by the integrity constraints. If any of the ICs is violated, the change is not performed. In this case the action ends by failure.

In the second group are concepts specifying *matchmaking queries* on groups or agents. These queries are transformed in SPARQL queries [16] and executed on the inferred model. For example, query asking for an agent which is member of the group `grp` and has the responder `resp` is translated in SPARQL query in the following form:

```
SELECT ?Agent WHERE {
  ?Agent isMemberOf grp .
  ?Agent hasResponder resp .
}
```

This query can also be translated to OWL-DL query of this form:

$$Query \sqsubseteq \ni isMemberOf.grp \sqcap \ni hasResponder.resp$$

On the other hand, more complicated queries require using variables and cannot be expressed in description logic. For example, if we need an agent which is connected to a data source, where both are in the same computational group, it can be written in SPARQL in the following way:

```
SELECT ?Agent WHERE {
  ?Agent hasInitiator ?Init .
  ?Init sendsTo ?Agent1 .
  ?Agent1 a DataSource .
  ?Agent isMemberOf ?Group .
  ?Agent1 isMemberOf ?Group .
  ?Group a CompGroup .
}
```

The third group of concepts contains concepts informing about *results* of actions or queries.

VI. Example

The role-based model allows introduction of a social logic in control of agents. So, an agent can directly exploit and manipulate the concepts and individuals of the DL social model (e.g. roles, groups) in fulfillment of their goals by communication with ROA. On the other hand, the agent undertakes by registering of a role to handle messages of certain types which are among the responders of the role. In this section we elaborate the computational MAS scenario using various learning methods which requires more complicated structure of 3 groups, computational group, simple learning group and evolutionary algorithm group.

There are seven agents in the MAS: the task manager, RBF network computing agent, data source, vector quantization agent, evolutionary algorithm agent and its supplementary agents, i.e. chromosome representation of an individual and tuner. The data classification problem by RBF network is solved by two different learning algorithms. The vector quantization determines the means of RBF units. In the next step other parameters of the model are optimized by the evolutionary algorithm. For the sake of simplicity we assume that these agents are started at the beginning.

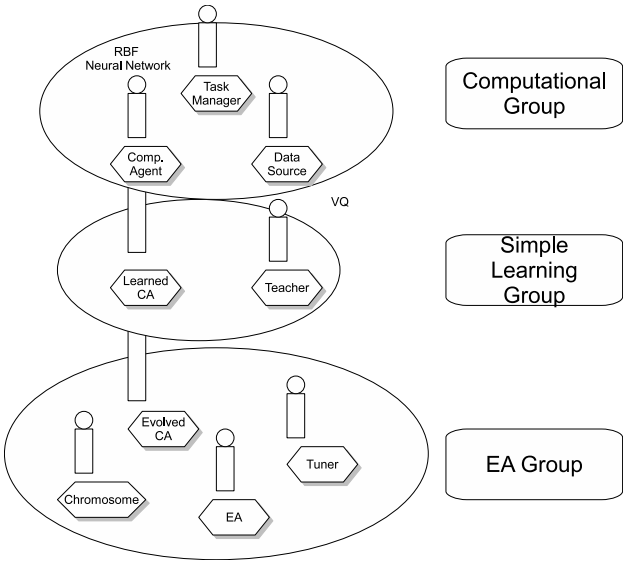

Fig. 5. Organization in a state of the computational MAS during computation.

Algorithm 1 Pseudocode of the task manager agent

1: Registration of the agent by ROA.
2: Set the $TaskManager$ role.
3: Create of group grp_1 with $CompGroup$ type.
4: Enter into grp_1.
5: Wait until the group grp_1 is prepared, i.e. all agents with corresponding roles are present.
6: Find any computational agent ca (agent with $CompAgent$ role) in the group grp_1.
7: Establish connection to ca, i.e. create initiator $init$ with type $ControlMsgInit$ and connect it to the agent ca.
8: **repeat**
9: Send a request message with description of the problem and random setting to ca.
10: Wait for a result.
11: **until** the problem is solved or the time exceeded
12: End the connection of the initiator $init$.
13: Leave the group grp_1.
14: Wait until the group grp_1 is empty.
15: Destroy the group grp_1.
16: Remove the $TaskManager$ role.

The *task manager* represents the problem which should be solved, and coordinates the actions of other agents in its computational group. A pseudocode of this agent is in Algorithm 1. The algorithms describe mainly organizational aspects of this scenario and consist of statements relating the DL model in ROA. The task manager sends control messages to the *RBF network* (Algorithm 2), i.e. a computational agent which should classify data. In order to do this, it needs training and testing data from a data source (*database agent* Algorithm 3).

The RBF network also creates two new groups with agents whose purpose is to learn its parameters. The *vector quantization algorithm* finds means of clusters in the training data (Algorithm 4).

The next step, evolutionary learning, is performed by an *evolutionary algorithm* (Algorithm 5) and its two supplementary agents. The *tuner* represents its settings. The

Algorithm 2 Algorithm of the RBF network

1: Registration of the agent by ROA and set the $CompAgent$ role.
2: Wait until a group with type $CompGroup$ is found. Name it grp_1. Enter into grp_1.
3: Wait until the group grp_1 is prepared, i.e. all agents with corresponding roles are present.
4: Find any data source agent ds (agent with $DataSource$ role) in the group grp_1.
5: Establish connection to ds, i.e. create initiator $init_1$ with type $DataMsgInit$ and connect it to the agent ds.
6: **while** the group is full **do**
7: **if** a request for computation (i.e. with the type $ControlMsg$) comes **then**
8: Send a request for a training data to the ds. Wait for the reply.
9: Set a $LearnedCA$ role.
10: Create and enter into group grp_2 of type $SimpleLearningGroup$.
11: Wait until there is an agent t with $Teacher$ role in the group grp_2.
12: Create initiator $init_2$ of type $LearningMsgInit$ and establish a connection to the agent t.
13: Send a request for finding means of clusters in the data to the agent t. Wait until this is finished.
14: End the connection of the initiator $init_2$.
15: Leave the group grp_2.
16: Set a $EvolvedCA$ role.
17: Create and enter into group grp_3 of type $EvoAlgorithmGroup$.
18: Wait until the group grp_3 is prepared.
19: Find an agent ea with $EvoAlgorithm$ role in the group grp_3.
20: Create initiator $init_3$ of type $LearningMsgInit$ and establish a connection to the agent ea.
21: Send a request for optimization of other parameters to the agent ea.
22: **repeat**
23: **if** a request of the type $ComputeModelMsg$ comes **then**
24: Evaluate the model with the received parameters and send back the results.
25: **end if**
26: **until** the ea finds a solution
27: End the connection of the initiator $init_3$.
28: Leave the group grp_3.
29: Wait until the group grp_2 is empty.
30: Destroy the group grp_2 and remove the role $LearnedCA$.
31: Wait until the group grp_3 is empty.
32: Destroy the group grp_3 and remove the role $EvolvedCA$.
33: Do the necessary computations.
34: Send a result of the computation as a reply to the request.
35: **end if**
36: **end while**
37: End the connection of the initiator $init_1$.
38: Leave the group grp_1.
39: Remove the $CompAgent$ role.

Algorithm 3 Pseudocode of the database agent

1: Registration of the agent by ROA and set the $DataSource$ role.
2: Wait until a group with type $CompGroup$ is found. Name it grp_1. Enter into grp_1.
3: Wait until the group grp_1 is prepared, i.e. all agents with corresponding roles are present.
4: **while** the group is full **do**
5: **if** a request of the type $DataMsg$ comes **then**
6: Find the data file according to the specification in the database.
7: Send the data as a reply to the request.
8: **end if**
9: **end while**
10: Leave the group grp_1.
11: Remove the $DataSource$ role.

chromosome translates individuals' representation into CA's models and computes fitnesses. After the processing of data and successful classification, the task manager is informed about its result. The developed state of the MAS is showed in Figure 5.

In this scenario, we have not taken into account possibilities of automatic creation and configuration of the system which can exploit the formal description of agents' capabilities. For example, there could be various chromosome representations of an evolved CA in the evolutionary algorithm which influences learning speed of the evolution. The creation of suitable agent type playing the chromosome role according to the evolved CA may be subject of metalearning. This is also the case of choice of the computational method with respect to the character of data.

Algorithm 4 Pseudocode of the vector quantization agent

1: Registration of the agent by ROA and set the $Teacher$ role.
2: Wait until a group with type $SimpleLearningGroup$ is found. Name it grp_1. Enter into grp_1.
3: Wait until the group grp_1 is prepared, i.e. both agents with corresponding roles are present.
4: **while** the group is full **do**
5: **if** a request of the type $LearningMsg$ comes **then**
6: Execute the learning process.
7: Send the reply.
8: **end if**
9: **end while**
10: Leave the group grp_1.
11: Remove the $CompAgent$ role.

Algorithm 5 Pseudocode of the evolutionary algorithm agent

1: Registration of the agent by ROA and set the $EvoAlgorithm$ role.
2: Wait until a group with type $EvoAlgorithmGroup$ is found. Name it grp_1. Enter into grp_1.
3: Wait until the group grp_1 is prepared.
4: Find an agent t with the role of $Tuner$.
5: Create initiator $init_1$ of type $EAParamsMsgInit$ and establish a connection to the agent t.
6: Find an agent ch with the role of $Chromosome$
7: Create initiator $init_2$ of type $FitnessMsgInit$ and establish a connection to the agent ch.
8: **while** the group is full **do**
9: **if** a request of the type $EAControlMsg$ comes **then**
10: Execute the evolutionary process, where parameters of the algorithm are obtained from the agent t and individual's fitness is computed by the agent ch.
11: Send the reply.
12: **end if**
13: **end while**
14: End the connections $init_1$ and $init_2$.
15: Leave the group grp_1.
16: Remove the $EvoAlgorithm$ role.

VII. Conclusion

In large and open multi-agent systems, where agents can join and leave different groups and behave according to its changing position in the environment, the emphasis shifts from the inner structure and algorithmic logic of individual agents to their interaction and cooperation aspects. Concept of role is a tool for modeling of MAS from this organizational perspective. An example of such a heterogeneous system is computational MAS, system supporting creation of hybrid intelligence model.

In this paper we have proposed a role-based model of computational MAS. We presented description logic formalization of this model and of its elementary blocks, roles, groups and interactions. The necessity of division of the model between open-world and closed-world rules has been highlighted. In the next step we implemented the ontology agent which exploits this model to manage MAS. The ontology agent keeps track the current state of the system, controls correctness of the system, and serves as a matchmaking middle agent.

These functionalities have been tested by implementation of computational MAS scenario. Even this simple example has proved that employing of social concepts simplifies not only the application development process but also the run-time control of agents. The expansion of the model to more complicated MAS is easy with respect to the modular character of the role model.

The future work will be put in improvement of the model in order better to cope with dynamic character of MAS. The groups in the system could be in different modalities, e.g. in the state of their creation in contrast to the complete state, where all agents with appropriate roles are present. The description of such states in the model would allow the agents better to detect the current situation of the environment. We will also implement and test other more sophisticated matchmaking algorithms which would take into account also a description of computational capabilities and previous results of the agent.

REFERENCES

[1] G. Weiss, Ed., *Multiagent Systems*. MIT Press, 1999.

[2] G. Cabri, L. Ferrari, and L. Leonardi, "Agent role-based collaboration and coordination: a survey about existing approaches," in *Proc. of the Man and Cybernetics Conf.*, 2004.

[3] P. Bonissone, "Soft computing: the convergence of emerging reasoning technologies," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, pp. 6–18, 1997.

[4] M. Wooldridge, N. R. Jennings, and D. Kinny, "The gaia methodology for agent-oriented analysis and design," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 3, pp. 285–312, 2000.

[5] G. Cabri, L. Ferrari, and L. Leonardi, "Supporting the development of multi-agent interactions via roles," in *AOSE 2005*, J. P. Mller and F. Zambonelli, Eds., no. LNCS 2935, 2006, pp. 154–166.

[6] J. Ferber, O. Gutknecht, and M. Fabien, "From agents to organizations: An organizational view of multi-agent systems," in *AOSE 2003*, P. Giorgini *et al.*, Eds., no. LNCS 3950, 2004, pp. 214–230.

[7] M. Sensoy, T. J. Norman, W. Vasconcelos, and K. Sycara, "OWL-POLAR: Semantic policies for agent reasoning," in *9th International Semantic Web Conference (ISWC2010)*, 2010, pp. 679–695.

[8] K. Sycara, "Multi-agent infrastructure, agent discovery, middle agents for web services and interoperation," in *Multi-agent systems and applications*, no. LNCS 2086. Springer Berlin / Heidelberg, 2006, pp. 17–49.

[9] C. Diamantini, D. Potena, and E. Storti, "KDDONTO: An ontology for discovery and composition of KDD algorithms," in *ECML/PKDD09 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*, 2009, pp. 13–24.

[10] G. Fortino and S. Galzarano, "Programming wireless body sensor network applications through agents," in *WOA*, 2010.

[11] Z. Zhang and C. Zhang, *Agent-Based Hybrid Intelligent Systems*. Springer Verlag, 2004.

[12] R. Neruda and G. Beuster, "Emerging hybrid computational models," in *Proc. of the ICIC 2006*, no. LNCS 4113, 2006, pp. 379–389.

[13] R. Neruda and O. K. k, "Role-based induced social norms in computational multi-agent systems," in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2011, WCECS 2011, 19-21 October*, 2011, pp. 437–442.

[14] R. Neruda and G. Beuster, "Toward dynamic generation of computational agents by means of logical descriptions," *International Transactions on Systems Science and Applications*, pp. 139–144, 2008.

[15] F. Baader *et al.*, *The description logic handbook: Theory, implementation, and applications*. Cambridge University Press, 2003.

[16] E. Sirin and J. Tao, "Towards integrity constraints in OWL," in *OWLED*, 2009.

[17] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C, Tech. Rep., 2006.

[18] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[19] K. Gibert, J. Izquierdo, G. Holmes, I. Athanasiadis, J. Comas, and M. Sanchez-Marre, "On the role of pre and post-processing in environmental data mining," in *International Congress on Environmental Modelling and Software – 4th Biennial Meeting*, 2008, pp. 1937–1958.

[20] O. Kazík and R. Neruda, "Role-based ontology model for management of data mining mas," in *Proc. of The Seventh International Workshop on Agents and Data Mining Interaction*, 2011, submitted.

[21] F. Bellifemine, G. Caire, and D. Greenwood, *Developing multi-agent systems with JADE*. John Wiley and Sons, 2007.