DISTRIBUTED COMPUTATION MULTI-AGENT SYSTEM

Maja ŠTULA, Darko STIPANIČEV, Josip MARAS

Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, R. Boskovica 32, 21000 Split, Croatia

{maja.stula, darko.stipanicev, josip.maras}@fesb.hr

Received Submission date: 29.08.2012.

Abstract This article addresses a formal model of a distributed computation multi-agent system. This model has evolved from the experimental research on using multi-agent systems as a ground for developing fuzzy cognitive maps. The main paper contribution is a distributed computation multi-agent system definition and mathematical formalization based on automata theory. This mathematical formalization is tested by developing distributed computation multi-agent systems for fuzzy cognitive maps and artificial neural networks – two typical distributed computation systems. Fuzzy cognitive maps are distributed computation systems used for qualitative modeling and behavior simulation, while artificial neural networks are used for modeling and simulating complex systems by creating a non-linear statistical data model. An artificial neural network encapsulates in its structure data patterns that are hidden in the data used to create the network. Both of these systems are well suited for formal model testing. We have used evolutionary incremental development as an agent design method which has shown to be a good approach to develop multi-agent systems according to the formal model of a distributed computation multi-agent system.

Keywords Multi-agent System, Formal Model of a Distributed Computation MAS, Fuzzy Cognitive Map, Artificial Neural Network, Finite State Machine

§1 Introduction

Agent technology is applied in a broad range of applications that vary in complexity: from relatively simple applications that use only individual agents (e.g. for filtering e-mail), to complex systems whose functionality is provided by a joint work of a large number of agents (e.g. air traffic control, data mining on the Internet, industrial production optimization, etc.)[1, 2, 3]. At first glance it seems that these diverse systems have nothing in common, but practice shows that it is possible to use agents as key abstractions in different software systems [4].

Due to its properties, agent technology is widely used in physically distributed systems [5, 6, 7, 8]. Our paper points out that computationally distributed systems can also benefit from agent technology. Distributed computation is a common property of a variety of different systems whether the system was explicitly designed to compute or not. Any distributed computation system with components exhibiting individual characteristics can benefit from (intelligent or dumb) agent based design and application [9] and there are case studies that show the usefulness of the approach [10]. However, as far as we know, there is no formal definition of DCMAS. By providing a formal definition we can more easily introduce agent-based reasoning to other distributed computation applications.

In this paper we give a formal definition of a DCMAS. Since agents in a DCMAS are defined with their states and transitions, as the basis for the formalization we have used a finite state machine. To build a distributed computation multi-agent system, multi-agent system designer has to identify states and transitions of agents.

We have used the formal DCMAS model on two case studies by converting a fuzzy cognitive map (FCM) and an artificial neural network (ANN) to multi-agent systems. FCM is a distributed computation system used for system qualitative modeling and behavior simulation [11]. The map consists of nodes representing system concepts connected with cause-effect relationships [12, 11]. Each concept is an entity for itself. Therefore, it is logical to calculate the behavior of the each concept individually while considering the influence from other concepts. Classic FCM calculates behavior of all concepts together, and it is difficult to insert concept special characteristics in the classic FCM. This restricts the method implementation to systems with concepts that do not exhibit individual characteristics. By introducing agent technology in the FCM, this problem could be resolved. In the obtained distributed computation multi-agent system called agent-based FCM (ABFCM), each FCM node is represented with an individual

agent. Agents autonomy allows adding individual characteristics to each concept represented with an agent.

Artificial neural networks (ANN) are also typical distributed computation systems. ANN is mimicking biological neural networks and it is created as an interconnected group of artificial neurons [29, 30]. It is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs [31]. Since a neural network is a massively parallel distributed processor it has a natural propensity for storing experiential knowledge and making it available for use [32]. Neural networks are usually used to model complex relationships between inputs and outputs and thus to find patterns in input data, or to capture the statistical structure in an unknown joint probability distribution between observed variables. Each neuron operates asynchronously on local information only. By introducing agent technology in artificial neural networks we again can insert special characteristics in each neuron (i.e. different threshold). The introduction of multi-agent technology to the other distributed computation systems could lead to similar effects of introducing new properties to the existing distributed computation systems like in cases of FCM or ANN.

The paper is organized as follows: section 2 gives the introduction to multi-agent systems, while section 3 describes the DCMAS, and presents the formalization of the concepts. Sections 4 and 5 in detail describe two case studies: simulating FCMs and ANNs with DCMAS respectively. Finally section 6 gives a conclusion and presents possible future work.

§2 Multi-agent Systems

An agent is a software or a hardware system situated in an environment. It can receive stimulus from the environment and flexibly and autonomously act in pursuing its goals [13, 3]. Agent environment can be anything from the physical world to the Internet, or in the case of a distributed computation multi-agent system – other agents involved in computation. The agent complexity is correlated with the environment properties because more complex environment implies more complex agent behavior to handle that environment. Designing a multi-agent system reduces down to designing an agent in that multi-agent system. The most important part of creating, designing or realizing an agent is identifying its behavior. Since that behavior highly depends on the environment, we also have to exactly identify the environment.

Table 1 Environment properties classification by Russel and Norvig

accessible vs.	inaccessible
deterministic vs.	nondeterministic
episodic vs.	non-episodic
static vs.	dynamic
discrete vs.	continuous

Russell and Norvig [14] environment properties classification is shown in Table 1. The environment is accessible if an agent can obtain complete, accurate and recent information about the environment. Usually, the environment is inaccessible to a degree. The lower the inaccessibility, the simpler the agent is, and vice versa. If an action, made by the agent, always has the same effect, the environment is considered deterministic. In an episodic environment the agent acts only on its current state, because there is no connection between agents previous, current or future actions. The static environment changes only under agent action. Other processes, that change the environment, exist in the dynamic environment. The environment will be discrete if it has certain, finite number of states and actions.

An agent can act without user intervention relying only on its own states and knowledge. This makes the agent autonomous. Agent flexibility is expressed with agent responsiveness, social ability and pro-activeness. The agent perceives its environment and reacts on the changes in the environment. It can act jointly with other agents or humans to accomplish a task or to help others in achieving a goal. Finally, an agent should exhibit goal-oriented behavior with taking the initiative to achieve its goal. In addition to aforementioned properties agents can have additional properties like mobility, learning capabilities, etc. [3].

Multi-agent system (MAS) can be defined as a network of entities working together on solving the problem that is beyond the agents individual solving capabilities and knowledge [3]. Each agent has a partial information or problem solving capability. Global system control does not exist in MAS. Information and knowledge needed for achieving a goal is decentralized. MAS is simply a group of agents, with afore-mentioned properties, that communicate [1]. Communication is the foundation of the multi-agent system, and agents need to share a common agent communication language (ACL) and a common ontology to be able to communicate [15]. A majority of existing multi-agent systems today rely

on the Foundation for Intelligent Physical Agents (FIPA) standards [16, 17]. FIPA is IEEE Computer Society standards organization dedicated to the promotion of technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings. FIPA has published, among others, ACL specification [18] and MAS abstract architecture specification [19].

In order to ensure FIPA specification compliance, all agents and multi-agent systems defined, formalized of described in this article are developed using JADE (Java Agent DEvelopment Framework). JADE is an open-source FIPA compliant multi-agent platform [20, 21].

§3 Distributed Computation Multi-agent Systems

In this paper, we formally define a distributed computation multi-agent system (DCMAS) as a system with a set of agents that each performs specified distributed computation. Each agent is defined by the environment the agent is situated in, and with its behavior while interacting with the respective environment. In the case of a distributed computation multi-agent system, the environment is limited to other agents involved in distributed computation. Therefore agent interaction with the environment is reduced to interactions with other agents. The behavior of each agent is a realization of a segment of the overall computation – in order to acquire the final result, the distributed computation has to be coordinated among agents.

An agent in DCMAS is defined with states and state transitions. We claim that an agent in a distributed computation can formally be defined as an automata (Definition 3.1.). Coordination is defined with agent state transition events. State transition events in each agent are dependent on state transition events in other agents participating in the same distributed computation and on the agent internal behavior set in agent states.

Definition 3.1
The DCMAS agent is mathematical defined as a 7-tuple:

$A_{DMAS} = (S, U, I, P, S_{in}, \omega, \delta)$, where:

- S is the finite non-empty set of agent's states, the union of sets S_{local} and $S_{external}$, $S = S_{local} \cup S_{external}$
- U is the finite input alphabet for A_{DMAS}, a set of agent ACL messages and defined agent internal symbols,

- I is the finite output alphabet for A_{DMAS}, a set of agent ACL messages and defined agent internal symbols,
- P is the finite set of agent's internal behavior conditions,
- S_{in} is the set of agent initial states, $S_{in} \subset S$,
- ω is the agent output function, $\omega : S \times (U \cup P) \rightarrow S$
- δ is the agent next-state transition function, $\delta : S \times (U \cup P) \rightarrow S$

Agents states that do not generate events affecting other agents are called local states (denoted with S_{local}), and in these states an agent has no influence on other agents, and does not generate outgoing ACL messages. An agent also has external states (denoted with $\mathrm{S}_{external}$), which are the states in which an agent influences other agents, and participates in the joint distributed computation by generating outgoing ACL messages. A local state can generate state transition events to external states, and vice-versa. This is why S_{local} and $\mathrm{S}_{external}$ are joined in one finite non-empty set of agent states denoted with S. The state dependencies between local and external states in one agent and between external states among agents are dynamically specified at run time from a set of predefined dependency types. Each agent stores in its local knowledge database definition of the transitions events for the local states, definition of the transitions events for the external states and the definition of the transition events between local and external states. Agent's states and transition events define distributed computation multi-agent system agents communication, coordination and cooperation, as well as agent internal behavior.

3.1 DCMAS agent identification

Agent internal symbols are used in the state transitions that are not caused by other agents. These symbols are stored in the local knowledge database of the agent, together with the ACL messages the agent can receive or send. Agent's internal behavior conditions, denoted with the P, are conditions that can occur during the execution of an agent behavior and can cause generation of the output symbol to other agents and/or state transition event in the observed agent. Internal behavior conditions are not caused by the state transition events from the observed agent or from other agents participating in the distributed computation. These conditions are defined within agent behavior without influence from other states or input symbols. For example, completion of a behavior is a typical internal behavior condition that can cause a state transition; or an error in the behavior.

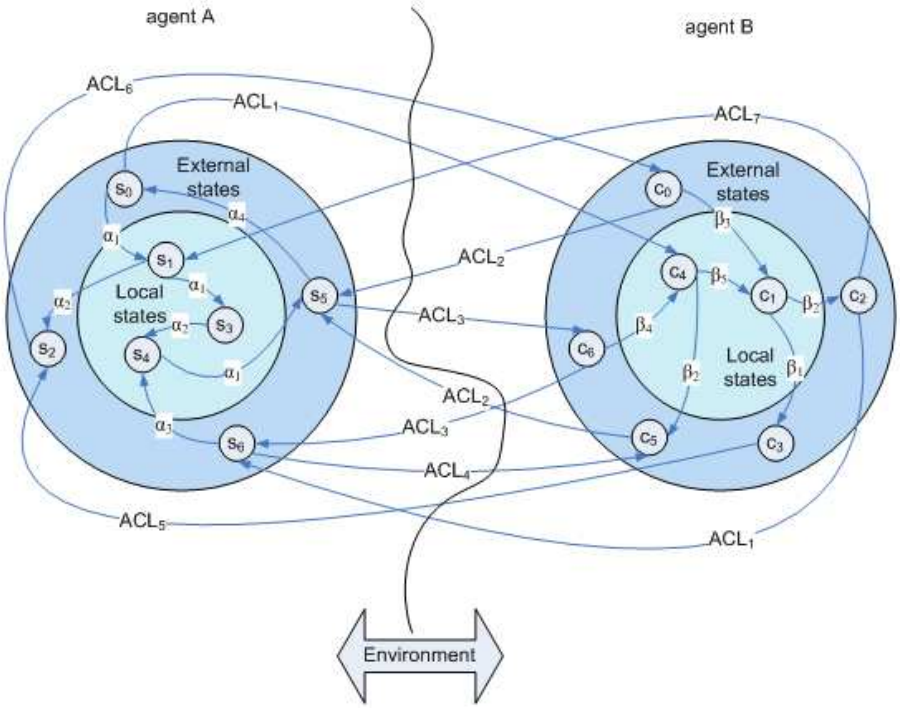

Fig. 1 The two agents of a distributed computation multi-agent system.

Figure 1 displays two agents in a distributed computation multi-agent system. Each agent has local and external states. External states generate outgoing ACL messages. An agent can transit from a local to an external state and from an external state to a local state. Local states can receive incoming ACL massage but cannot send outgoing ACL messages. The state transitions can be triggered by the ACL message, when agent receives stimulus from the environment and acts affecting the environment by sending the ACL message. The local transition events are defined with the generation of the agent internal symbols or with internal behavior conditions within agent internal behavior since local transition events are not triggered by other agents participating in the same distributed computation. All transition events are stored in the local knowledge database in each of the agents.

Agent's internal behavior conditions are defined, but are not shown in the Figure 1 due to the complexity of the graphical display. According to the distributed computation multi-agent system agent definition, agent A is a 7-tuple: $(S_A, U_A, I_A, P_A, S_{inA}, \omega_A, \delta_A)$ where:

- $S_A = S_{localA} \cup S_{externalA}, S_{localA} = \{s_1, s_3, s_4\}, S_{externalA} = \{s_0, s_2, s_5, s_6\}$
- $U_A = \{ACL_1, ACL_2, ACL_3, ACL_5, ACL_7, \alpha_1, \alpha_2, \alpha_3, \alpha_4\}$
- $I_A = \{ACL_1, ACL_3, ACL_4, ACL_6, \alpha_1, \alpha_2, \alpha_3, \alpha_4\}$
- $P_A = \{error, end\}$
- $S_{inA} = \{s_0\}$
- ω_A is the agent output function, $\omega_A : S_A \times (U_A \cup P_A) \rightarrow I_A$
- δ_A is the agent next-state transition function, $\delta_A : S_A \times (U_A \cup P_A) \rightarrow S_A$

Agent B is defined as a 7-tuple $(S_B, U_B, I_B, P_B, S_{inB}, \omega_B, \delta_B)$ where:

- $S_B = S_{localB} \cup S_{externalB}, S_{localB} = \{c_1, c_4\}, S_{externalB} = \{c_0, c_2, c_3, c_5, c_6\}$
- $U_B = \{ACL_1, ACL_3, ACL_4, ACL_5, ACL_6, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$
- $I_B = \{ACL_1, ACL_2, ACL_3, ACL_5, ACL_7, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$
- $P_B = \{error, start, end\}$
- $S_{inB} = \{c_0\}$
- ω_B is the agent output function, $\omega_B : S_B \times (U_B \cup P_B) \rightarrow I_B$
- δ_B is an agent next-state transition function, $\delta_B : S_B \times (U_B \cup P_B) \rightarrow S_B$

A DCMAS agent design, with the proposed formalization, reduces to the definition of local and external states, agent internal symbols, agent's behavior conditions and ACL messages exchanged between the agents participating in the same distributed computation. That knowledge is stored in the local knowledge database in each of the agents. The external state is defined as a state in which an agent can create and send ACL messages to other agents and can receive ACL messages. The local state is a state in which an agent can not send ACL messages to the other agents, but can receive ACL messages. Agent internal symbols are local for one agent and do not affect other agents. Also the agent's internal behavior conditions are local for one agent. The behavior conditions are not caused by any state transition, but they cause state transition and generate output alphabet symbols.

3.2 DCMAS agent equality

Frequently, agents in a distributed computation multi-agent system are the same. The next definition formalizes the equality of two DCMAS agents.

Definition 3.2
Two distributed computation multi-agent system agents, A defined with a 7-tuple $(S_A, U_A, I_A, P_A, S_{inA}, \omega_A, \delta_A)$, and B defined with a 7-tuple:
$(S_B, U_B, I_B, P_B, S_{inB}, \omega_B, \delta_B)$, are the same if:

- $S_{localA} = S_{localB}$ & $S_{externalA} = S_{externalB}$
- $U_A = U_B$
- $I_A = I_B$
- $P_A = P_B$
- $S_{inA} = S_{inB}$
- $\omega_A = \omega_B$
- $\delta_A = \delta_B$

In the following chapters we explain in detail how we have used this DCMAS definition to design and develop multi-agent systems for distributed computation by using two case studies: fuzzy cognitive maps and artificial neural networks.

§4 Test cases for DCMAS

We have selected two distributed computation systems which we have prior experience with: fuzzy cognitive map (FCM) and artificial neural network, and we have designed and developed multi-agent systems according to the definition of a distributed computation multi-agent system from chapter three. Designed multi-agent systems completely mimic fuzzy cognitive map and neural network producing the same results as a classic implementation of these distributed computation systems. This raises the question why bother to build multi-agent systems for any distributed computation system if it behaves exactly the same. The answer is that agent autonomy enables introducing new properties to existing distributed computation system that are hard to use in a non multi-agent distributed computation system. For example, in FCM we can use a different inference algorithm in each node. Similarly, each neural network node can, for example, use a different threshold. Listed are simple differences between distributed computation entities but there is no limitation, since agents intrinsically posses higher level of autonomy then entities in non multi-agent distributed computation systems [3, 13].

4.1 Fuzzy Cognitive Map

Fuzzy Cognitive Map is a qualitative modeling method and system dynamic behavior technique that represents a system as a group of concepts and cause-effect relations among those concepts [11]. Concepts and cause-effects values must be mapped to the [-1, 1] interval. The larger the cause-effect relation weighted factor, the stronger the cause-effect relation among concepts. The pos-

itive weighted factor sign indicates that if the concept from which cause-effect relation originates increases, then the concept in which cause-effect relation terminates also increases. The negative weighted factor sign indicates that if the concept from which cause-effect relation originates increases, then the concept in which cause-effect relation terminates decreases. In its simplest form the FCM inference process is a manipulation of two matrices, the concept vector and the adjacency matrix [12, 22]. Concept vector contains FCM nodes (concepts) values, while the adjacency matrix carries cause-effects values among nodes. If A_i^t is the value of *i-th* node at the discrete time step t, and w_{ji} is *i-th* column element in the adjacency matrix, then A_i^t is calculated according to the:

$$A_i^t = f(\sum_{j=1, j\neq i}^{n} A_j^{t-1} w_{ji}) \qquad (1)$$

A_j^{t-1} is the value of *j-th* node at the discrete time step $t-1$; f is transformation function, used to normalize the value of a node to the interval [-1, 1]; n is the total number of map nodes. The conclusion carried out with the FCM can be made when the map reaches its limited cycle. In a map limited cycle the node values are repeating. The conclusion can also be made when the map reaches its fixed point, and the node values remain the same. In the case of the chaotic map, an accurate conclusion cannot be made. A Fuzzy Cognitive Map is a combination of a conceptual map and fuzzy logic which provides a more realistic and more accurate real word portrait than the conceptual maps binary logic [23].

[1] Design and development of FCM distributed computation multi-agent system

The FCM distributed computation multi-agent system (named Agent-Based Fuzzy Cognitive Map (ABFCM) [24] is obtained with the one-way mapping function from the FCM concepts set to the agent set. Each concept is translated into exactly one agent.

An ABFCM behaves identically with a classic FCM if the system observed by means of a FCM model operates in such a way that the concepts do not require the different inference algorithm. The process of designing the distributed computation multi-agent system used as a FCM is explained in detail. A DCMAS agent design, with the proposed formalization, reduces to the definition of local and external states, agent internal symbols, agent's behavior conditions and ACL messages exchanged between the agents participating in

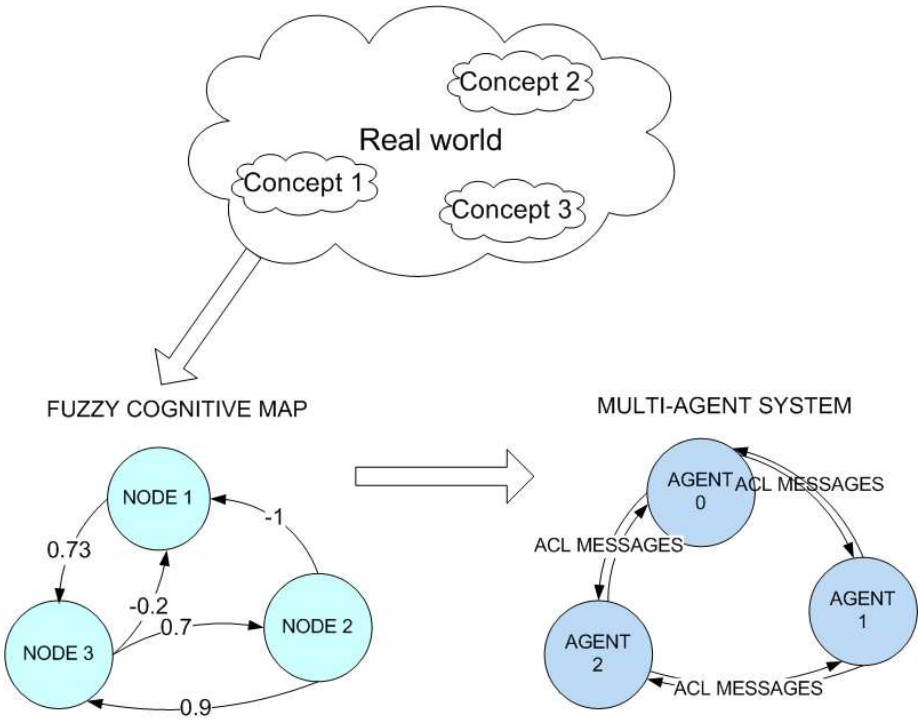

Fig. 2 Creating a Fuzzy Cognitive Map from the real world and translating it into the Agent-Based Fuzzy Cognitive Map

the same distributed computation. The knowledge about external states, local states, agent internal symbols, internal behavior conditions and ACL messages exchanged between the agents participating in the same distributed computation is stored in the local knowledge database in each of the agents.

Agents in the ABFCM have a subsumption [25] architecture that does not require symbolic representation of agent's knowledge. Agent behavior is broken down in to a hierarchy of simple behaviors. According to Brooks, each layer contains just a part of agent behavior and overall agent intelligence is manifested through the interaction of all layers and through agent interaction with the environment. Subsumption architecture applied to the agent is similar to hybrid horizontal layer architecture (shown in Fig. 3) [26].

Thereby, ABFCM agent architecture will be viewed as horizontal layered architecture. We refer to Brooks architecture to de-emphasize the need for symbolic knowledge representation. Nonsymbolic knowledge representation considerably facilitates the development of the multi-agent system and is usually

used in MAS. As shown in Figure 3, there are five different simple agent behaviors (*StartActiveBehavior*, *WaitBehavior*, *DieBehavior*, *ActiveBehavior* and *QueryBehavior*) layered into overall agent behavior.

The horizontal layered architecture is chosen during the ABFCM agents design because it was difficult to isolate all possible agent actions at the beginning of the ABFCM design. In each step of the ABFCM multi-agent system development, the segment of the needed and desired agent behavior has been identified and realized. Since the agent behaviors are introduced gradually in the multi-agent system, agent horizontal layered architecture was appropriate. The layered architecture made it possible to add new layer of behavior, when the new behavior was identified, without major change in the existing agent behaviors. This is done using the evolutionary incremental development of the multi-agent system [27].

The JADE agent platform agent behavior prototype class, used to realize ABFCM agents behavior class (*FCMAgentbehavior*), models agent behavior as finite state machine. Each machine state is realized as one of the five behaviors identified during the design phase. The chosen agent behavior prototype class best meets the distributed computation multi-agent system definition. In the distributed computation multi-agent system, each agent is observed as a state machine with the states and transition events mutually conditioned and dependent on the states and transition events of other agents participating in the distributed computation.

The JADE agent behavior prototype that already models agent behavior as finite state machine was used as a foundation for the developed system. The main behavior *FCMAgentbehavior* defines finite state machine and can be

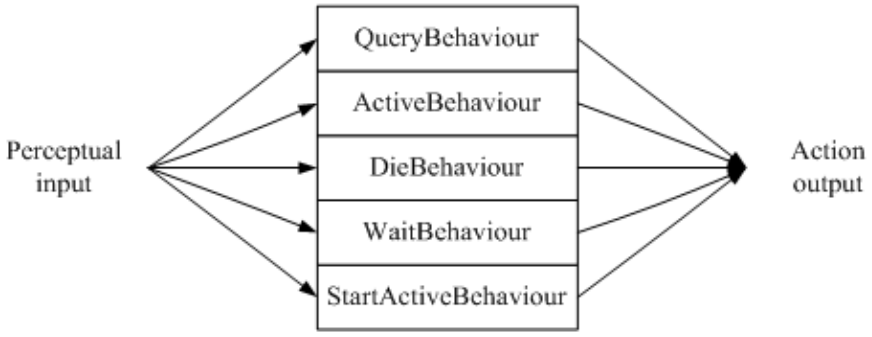

Fig. 3 ABFCM agent horizontal layered architecture

observed as mediator for the layered behaviors shown in Figure 3. It defines the five layered behaviors as one of the states and also defines possible state transition events. The *FCMAgentbehavior* behavior also provides the basic agent functionality like registration to the agent platform, starting and terminating agent life cycle, etc. [28]. After starting the agent, control of the agent overall behavior takes over *StartActiveBehavior* (START_STATE). The agent immediately transits to *WaitBehavior* (WAIT_STATE), *DieBehavior* (DIE_STATE), *ActiveBehavior* (ACTIVE_STATE) or *QueryBehavior* (QUERY_STATE) after starting, depending on the state transition event (Figure 4). When an agent is in one of the five possible states, that behavior has the control of agent overall behavior and takes an action if necessary. The agent has knowledge about the initial state of the environment when it starts. The initial state of the environment is the state of all agents that are related to the agent in stake.

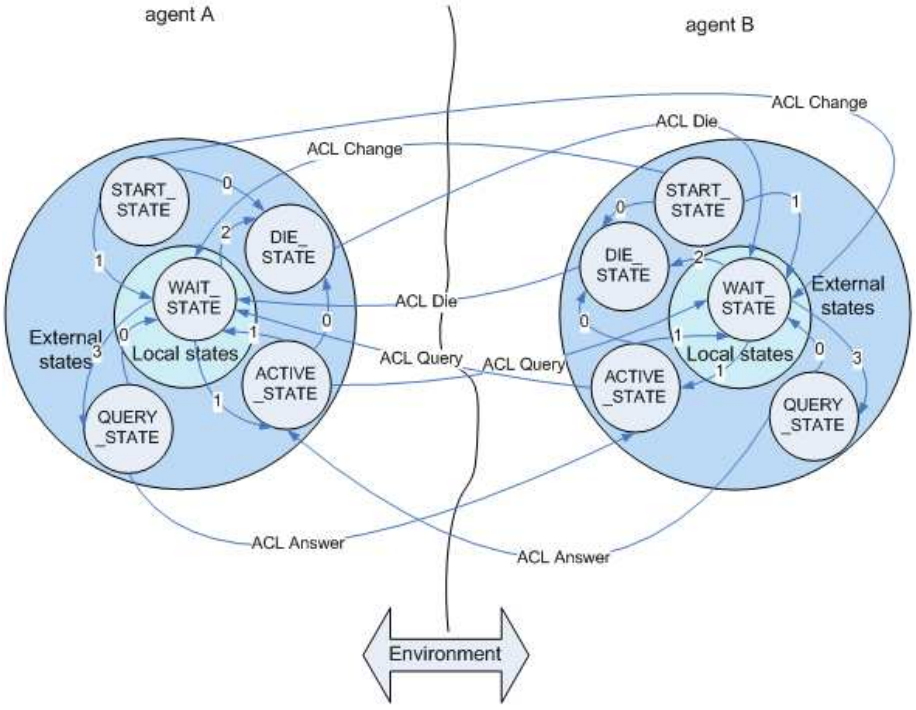

Fig. 4 The two ABFCM distributed computation multi-agent system agent's states and state transitions

The *StartActiveBehavior* uses that knowledge and generates the initial agent action without waiting for the input from the agent environment. The ini-

tial agent action is generation of the initial ACL Change message. The message generation is one of the agent's behavior conditions in the START_STATE. After that, behavior ends and agent transits to the *WaitBehavior* (WAIT_STATE) if there is no error (state transition marked 1). The local state transition event from the *StartActiveBehavior* to the *WaitBehavior* is the *StartActiveBehavior* error free completion. So, the error free behavior completion is one of the agent's behavior conditions. Local events are not triggered by other agents. A local event is a result of the agent internal behavior while an external event is a result of the actions, states and transitions outside the observed agent, i.e. in the agent environment. If an error occurred in the *StartActiveBehavior* behavior the error is processed as the state transition event, marked 0, to the *DieBehavior* behavior (DIE_STATE). So, the error is one of the agent's behavior conditions. The *DieBehavior* behavior terminates agent execution (i.e. agent life cycle) sending Die ACL message.

Ontology is a "conditio sine qua non" for a multi-agent system, thereby, at the same time that agent was designed, appropriate ontology was also developed. Ontology entities are identified by analyzing the problem domain and the multi-agent system designed to exhibit FCM behavior. When a new entity is identified, it is incrementally added to the ontology. Thus, the ABFCM ontology is completed. The term complete refers to the fact that developed ontology covers all necessary agent communication and knowledge exchange. It does not mean that developed ontology could not be expanded with more domain terms. The defined ABFCM ontology contains the four agent actions:

- *Change* is the action that the agent uses to inform other agents about changes of the node that agent is representing.
- *Answer* is the action that the agent uses to answer queries about his value.
- *Query* is the action that enables the agent to query other agents about their state.
- *Die* action is the action that agents use to agree on stopping the agent map.

An agent enters the *WaitBehavior* state if nothing is going on in the agent (e.g. new state calculation), or around the agent in the environment. Since the agent is modeled to exhibit FCM node behavior, the main agent action is the calculation of a new state. For the new node value calculation, the agent enters ACTIVE_STATE (*ActiveBehavior* behavior). In ACTIVE_STATE

the agent needs to get current values from other agents. So, it generates ACL *Query* messages to other agents. The *message generation* is one of the agent's internal behavior conditions in the ACTIVE_STATE causing the generation of the ACL Query message to other agents. State transition from the *WaitBehavior* to the *ActiveBehavior* is triggered by an external event generated by other agents participating in the ABFCM. The state transition event is the reception of the ACL message with the *Change* action in the message content. The state transition event from the WAIT_STATE to the QUERY_STATE (*QueryBehavior* behavior) is a reception of the ACL message with *Query* action in the message content.

The agent enters *QueryBehavior* behavior to answer other agents that are inquiring about the node value that the agent is representing. The *message generation* is one of the agent's internal behavior conditions in the QUERY_STATE causing the generation of the ACL *Answer* message to other agents. The completion of the *QueryBehavior* behavior is the state transition event, marked 0, transiting agent back to the WAIT_STATE. The local state transition event from the *QueryBehavior* to the *WaitBehavior* is the *QueryBehavior* error free completion. So, the *error free behavior completion* is also the agent's behavior condition for the QUERY_STATE. When an agent is in the WAIT_STATE it does not generate ACL messages to the other agents participating in the same distributed computation. This puts WAIT_STATE in the agent local states.

An agent generates and sends ACL messages in all other states, so all other states are agent external states. The process of creating the agent is finished when all local states, external states, input and output symbols, agent's behavior conditions are identified and output and agent next-state transition functions are defined.

If an error has occurred in the *ActiveBehavior* behavior, the error is processed as the state transition event, marked 0, to the *DieBehavior* behavior (DIE_STATE). Beside the error in the agent behavior, the agent life cycle ends when the map reaches its limited cycle or its fixed point. The detection of the limit cycle or fixed point is negotiated among agents using ACL messages *Die*. The state transition event to the *DieBehavior* is an agreement of all agents that the map has reached its limited cycle or fixed point after negotiation. Actions carried out in the *StartActiveBehavior* behavior can change the current state of the environment, resulting with inputs to the agent that can trigger the agent's state transition from the *WaitBehavior* behavior to the *ActiveBehavior* be-

havior. Those actions can result in more environment changes and consequently in more inputs to the agent generating the state transition events and transition to another behavior. Described actions and state transitions go on during the entire agent life cycle. In a nutshell, we have identified, using evolutionary incremental development, all local and external states, agent internal symbols, agent's behavior conditions and ACL messages and according to Definition, 1 we have defined a fuzzy cognitive map distributed computation multi-agent system agent.

The ABFCM distributed computation multi-agent system agent is a 7-tuple:

$ABFCM_{DMAS} = (S, U, I, P, S_{in}, \omega, \delta)$, where:

- $S = S_{local} \cup S_{external}, S_{local} = \{$ WAIT_STATE $\}, S_{external} = \{$START_STATE, QUERY_STATE, ACTIVE_STATE, DIE_STATE$\}$
- $U = \{Answer, Query, Change, Die, 0, 1, 2, 3\}$
- $I = \{Answer, Query, Change, Die, 0, 1, 2, 3\}$
- $P = \{error\ free\ behavior\ completion, error, message\ generation\}$
- $S_{in} = \{$START_STATE$\}$
- ω is the agent output function, $\omega : S \times (U \cup P) \to I$
  $\omega($START_STATE$, error) = 0$
  $\omega($START_STATE$, message\ generation) = Change$
  $\omega($START_STATE$, error\ free\ behavior\ completion) = 1$
  $\omega($WAIT_STATE$, Change) = 1$
  $\omega($WAIT_STATE$, Die) = 2$
  $\omega($WAIT_STATE$, error) = 2$
  $\omega($WAIT_STATE$, Query) = 3$
  $\omega($ACTIVE_STATE$, error) = 0$
  $\omega($ACTIVE_STATE$, error\ free\ behavior\ completion) = 1$
  $\omega($ACTIVE_STATE$, message\ generation) = Query$
  $\omega($QUERY_STATE$, error\ free\ behavior\ completion) = 0$
  $\omega($QUERY_STATE$, message\ generation) = Answer$
  $\omega($DIE_STATE$, message\ generation) = Die$

- δ is the agent next-state transition function, $\delta : S \times (U \cup P) \to S$
  $\delta($START_STATE,$error) =$ DIE_STATE
  $\delta($START_STATE,$error\ free\ behavior\ completion) =$ WAIT_STATE
  $\delta($WAIT_STATE,$Change) =$ACTIVE_STATE

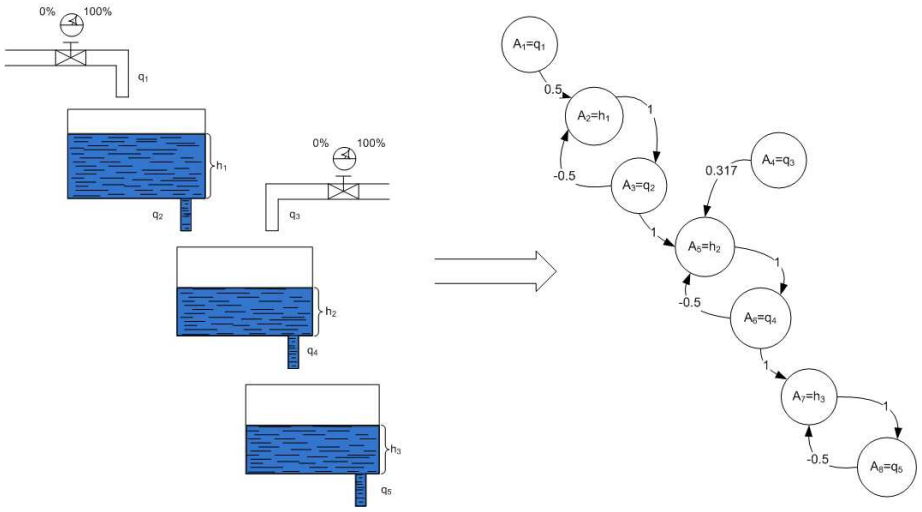

Fig. 5 On the left – three coupled tanks system b) The developed FCM map

δ(WAIT_STATE,Die) =DIE_STATE
δ(WAIT_STATE,$error$) =DIE_STATE
δ(WAIT_STATE,$Query$) =QUERY_STATE
δ(ACTIVE_STATE,$error$) =DIE_STATE
δ(ACTIVE_STATE,$error\ free\ behavior\ completion$) =WAIT_STATE
δ(QUERY_STATE,$error\ free\ behavior\ completion$) =WAIT_STATE

Experimental results: To test the developed multi-agent system we use the system shown in Figure 5. We have done testing to find out whether the obtained system behaves like the classic FCM, but we have used different inference algorithms in different map nodes – something that is hard to do in a classic FCM, while relatively simple in a multi-agent system. The map is usable if the simulation results are correct i.e. if the simulation results correspond with the real system behavior. We compared the simulation results obtained with the distributed computation multi-agent system fuzzy cognitive map with the Matlab model of the system.

There are eight main concepts in the nonlinear three coupled tanks system:

- A_1 – first tank input flow q_1
- A_2 – first tank flow height h_1
- A_3 – first tank outflow and second tank input flow q_2

Table 2 ABFCM results with the starting concept vector $A^t = 0 = [0.3000.50000]$.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Start state (k = 0)	0.3	0.0	0.0	0.5	0.0	0.0	0.0	0.0
k = 1	0.3	0.1488	0.0	0.5	0.1571	0.0	0.0	0.0
k = 2	0.3	0.1488	0.1477	0.5	0.1571	0.1558	0.0	0.0
...	...	...	...	...	...	...	...	...
k = 34	0.3	0.0999	0.0995	0.5	0.1714	0.1697	0.1082	0.1077
End state (k=35)	0.3	0.0999	0.0995	0.5	0.1714	0.1697	0.1082	0.1077

- A_4 – second tank input flow q_3
- A_5 – second tank flow height h_2
- A_6 – second tank outflow and third tank input flow q_4
- A_7 – third tank flow height h_3
- A_8 – third tank outflow q_5

Cause-effect relationships weights strongly depend on the system's physical values. The final system FCM is shown in Figure 5. Once the map is adjusted, the real tests are done and results are presented here. More details on identifying the map concepts and cause-effect relationships, as well as maps results can be found in [24].

In this paper we have shown a limited set of results, in order to demonstrate that the developed distributed multi-agent system behaves like a FCM. The new property of the developed distributed multi-agent system map, compared with the classic FCM, is the possibility of introducing different inference algorithm (Eq.1) in each map node. The classic FCM restricts all nodes to the same inference process. A system like a nonlinear three-coupled tanks system requires different inference algorithms in nodes and this is why it was chosen. The main advantage of basing distributed computation systems on a multi-agent approach is higher autonomy of computation components. In the nonlinear three-coupled tanks system, the difference in inference algorithm for input flows and outflows and flow heights is exhibited with different transformation functions. Nodes representing flows require a transformation function defined with Eq.2 while nodes representing flow heights use a transformation function shown in Eq.3. In a classic FCM, it is very hard to use different transformation functions in the map nodes.

$$(2) \quad f(x) = \begin{cases} -1 \times tanh(x), & tanh(x) \leq 0 \\ tanh(x), & tanh(x) > 0 \end{cases}$$

$$(3) \quad f(x) = tanh(x)$$

System behavior simulation with the initial values for the input flow value q_1 of 30% of the $25m^3/sec$, and input flow value q_3 of 50% of the $25m^3/sec$ is done by setting the concepts $A_1 = 0.3$ and $A_3 = 0.5$. Table 2 presents results obtained with the ABFCM. The obtained results require additional user interpretation that can also be found in [24].

4.2 Artificial Neural Network

An artificial neural network is composed of a very large number of simple processing elements behaving like biological neurons. An element receives stimulus from other elements, or from external sources, and acts according to the received stimulus and inference algorithm similar to that in a FCM (Figure 6).

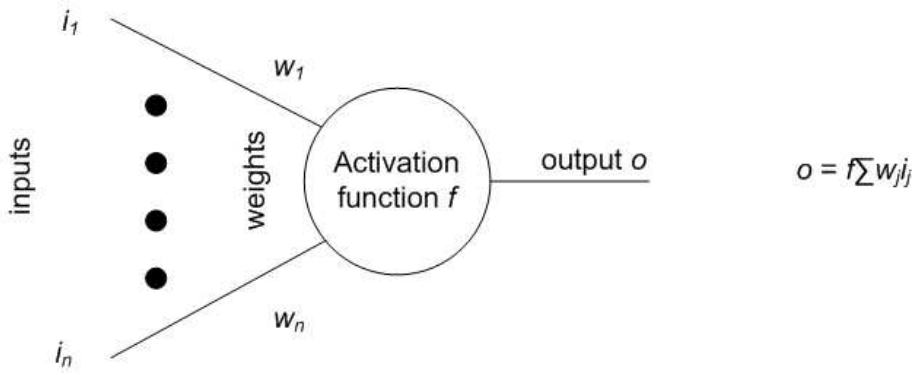

Fig. 6 Single neuron simple inference algorithm.

Elements are organized hierarchically in interconnected layers (Figure 7). Input layer receives external stimulus, and output layer generates an answer to imposed external stimulus. It is extremely rare that network has not at last one hidden layer between input and output, and often number of hidden layers is higher. Each element operates only on local information asynchronously without overall system clock [32].

Application scope of neural networks ranges from pattern recognition [33], optimization, to system control [34].

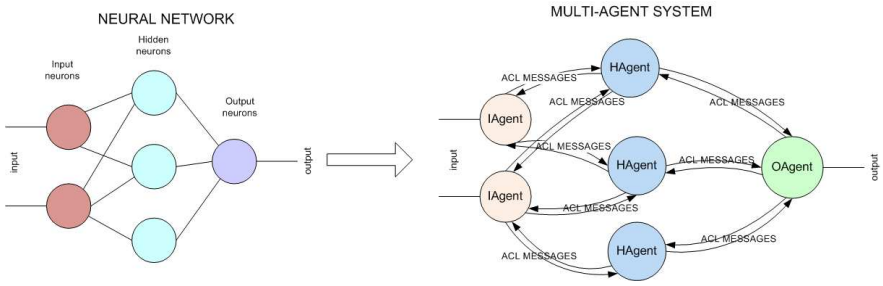

Fig. 7 Translating an artificial neural network into the ANN distributed computation multi-agent system.

[1] Design and development of ANN distributed computation multi-agent system

We have designed ABANN (Agent-Based Artificial Neural Network) for the simple feed-forward ANN when any output of the neurons is input to the neurons of next level. Each neuron is translated into one agent. In the case of ANN three types of DCMAS agents are identified: *i)* input ($ABANNI$), *ii)* hidden ($ABANNH$) and *iii)* output ($ABANNO$) matching appropriate neurons, since these three types of neurons differ in environment (Figure 8). The ANN input layer, consists of neurons that receive input from the user so for input neurons the environment consists of user enforced input and neurons of the next level. The hidden layer consists of neurons that receive input only from other neurons and generate output only to other neurons so agents communicate only with each other so for hidden neurons environment are preceding and following neurons. For output neurons environment are preceding neurons and user that requires network output information. Again the horizontal layered architecture is used during evolutionary incremental design of the multi-agent system that will be developed on the JADE agent platform.

The ABANN distributed computation multi-agent system agents are:

- a 7-tuple $ABANNI_{DMAS} = (S, U, I, P, S_{in}, \omega, \delta)$, where
  - $S = S_{local} \cup S_{external}, S_{local} = \{START\_STATE_IN\}, S_{external} = \{ACTIVE\_STATE, DIE\_STATE\}$
  - $U = \{Userinput, Change, 0, 1, 2\}$
  - $I = \{Die, 0, 1, 2\}$
  - $P = \{$error free behavior completion, error, message generation$\}$
  - $S_{in} = \{START\_STATE\_IN\}$
  - ω is the agent output function, $\omega : S \times (U \cup P) \rightarrow I$

$\omega(START\_STATE\_IN, error) = 0$
$\omega(START\_STATE\_IN, Userinput) = 1$
$\omega(START\_STATE\_IN,$ error free behavior$) = 2$
$\omega(START\_STATE\_IN, Die) = 0$
$\omega(ACTIVE\_STATE, Die) = 0$
$\omega(ACTIVE\_STATE, error) = 0$
$\omega(ACTIVE\_STATE,$ error free behavior completion$) = 1$
$\omega(ACTIVE\_STATE,$ message generation$) = Change$
$\omega(DIE\_STATE,$ message generation$) = Die$

- δ is the agent next state transition function, $\delta : S \times (U \cup P) \rightarrow S$
  $\delta(START\_STATE\_IN, error) = DIE\_STATE$
  $\delta(START\_STATE\_IN, Die) = DIE\_STATE$
  $\delta(ACTIVE\_STATE, Die) = DIE\_STATE$
  $\delta(ACTIVE\_STATE, error) = DIE\_STATE$
  $\delta(START\_STATE\_IN,$ error free behavior completion$) = START_STATE_IN$
  $\delta(START\_STATE\_IN,$ User Input$) = ACTIVE_STATE_IN$
  $\delta(ACTIVE\_STATE,$ error free behavior completion$) = START_STATE_IN$

- a 7-tuple $ABANNH_{DMAS} = (S, U, I, P, S_{in}, \omega, \delta)$, where
  - $S = S_{local} \cup S_{external}, S_{local} =$ { START_STATE}, $S_{external} =$ {ACTIVE_STATE, DIE_STATE}
  - $U = \{Change, 0, 1, 2\}$
  - $I = \{Change, Die, 0, 1, 2\}$
  - $P = \{errorfreebehaviorcompletion, error, messagegeneration\}$
  - $S_{in} =$ {START_STATE}
  - ω is the agent output function, $\omega : S \times (U \cup P) \rightarrow I$
    $\omega(START\_STATE, error) = 0$
    $\omega(START\_STATE, Change) = 1$
    $\omega(START\_STATE,$ error free behavior$) = 2$
    $\omega(START\_STATE, Die) = 0$
    $\omega(ACTIVE\_STATE, Die) = 0$
    $\omega(ACTIVE\_STATE, error) = 0$
    $\omega(ACTIVE\_STATE,$ error free behavior completion$) = 1$
    $\omega(ACTIVE\_STATE,$ message generation$) = Change$
    $\omega(DIE\_STATE,$ message generation$) = Die$

- δ is the agent next state transition function, $\delta : S \times (U \cup P) \to S$
  $\delta(START\_STATE, error) = DIE\_STATE$
  $\delta(ACTIVE\_STATE, error) = DIE\_STATE$
  $\delta(START\_STATE, Die) = DIE\_STATE$
  $\delta(ACTIVE\_STATE, Die) = DIE\_STATE$
  $\delta(START\_STATE,$ error free behavior completion$) = START_STATE$
  $\delta(START\_STATE, Change) = ACTIVE\_STATE$
  $\delta(ACTIVE\_STATE,$ error free behavior completion$) = START_STATE$

- a 7-tuple $ABANNO_{DMAS} = (S, U, I, P, S_{in}, \omega, \delta)$, where
  - $S = S_{local} \cup S_{external}, S_{local} =$ { START_STATE, ACTIVE_STATE_OUT, CREATE_OUTPUT}, $S_{external} =$ {DIE_STATE}
  - $U = \{Change, 0, 1, 2\}$
  - $I =$ {User output, Die, 0, 1, 2}
  - $P =$ {error free behavior completion, error, message generation, output generation}
  - $S_{in} =$ {START_STATE}
  - ω is the agent output function, $\omega : S \times (U \cup P) \to I$
    $\omega(START\_STATE, error) = 0$
    $\omega(START\_STATE, Change) = 1$
    $\omega(START\_STATE,$ error free behavior$) = 2$
    $\omega(START\_STATE, Die) = 0$
    $\omega(ACTIVE\_STATE\_OUT, Die) = 0$
    $\omega(CREATE\_OUTPUT, Die) = 0$
    $\omega(ACTIVE\_STATE\_OUT, error) = 0$
    $\omega(ACTIVE\_STATE\_OUT,$ error free behavior completion$) = 1$
    $\omega(CREATE\_OUTPUT, error) = 0$
    $\omega(CREATE\_OUTPUT,$ error free behavior completion$) = 1$
    $\omega(CREATE\_OUTPUT,$ output generation$) =$User output
    $\omega(DIE\_STATE,$ message generation$) = Die$

  - δ is the agent next state transition function, $\delta : S \times (U \cup P) \to S$
    $\delta(START\_STATE, error) = DIE\_STATE$
    $\delta(ACTIVE\_STATE\_OUT, error) = DIE\_STATE$
    $\omega(CREATE\_OUTPUT, error) = DIE\_STATE$

$\delta(START\_STATE, Die) = DIE\_STATE$
$\delta(ACTIVE\_STATE\_OUT, Die) = DIE\_STATE$
$\omega(CREATE\_OUTPUT, Die) = DIE\_STATE\_OUT$
$\delta(START\_STATE,$ error free behavior completion$) = START_STATE$
$\delta(START\_STATE, Change) = ACTIVE\_STATE\_OUT$
$\delta(ACTIVE\_STATE\_OUT,$ error free behavior completion$) =$
$CREATE\_OUTPUT$
$\delta(CREATE\_OUTPUT,$ error free behavior completion$) =$
$START\_STATE$

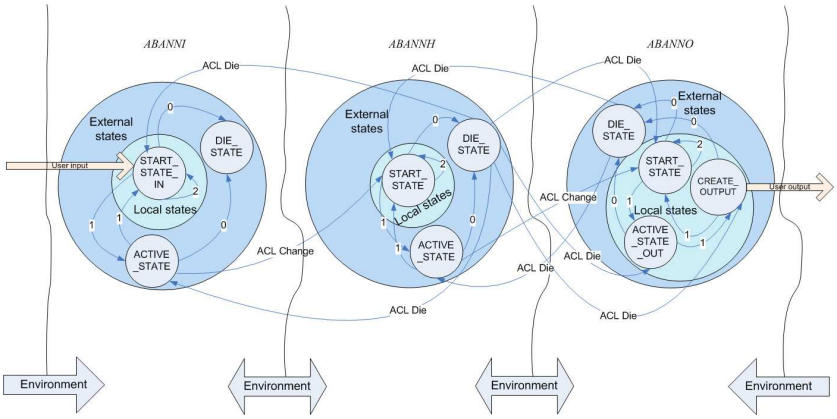

Fig. 8 The three ABANN agent types' states and state transitions.

After starting the agent, the control of the agent overall behavior is taken over by the *StartActiveInBehavior* (START_STATE_IN) for $ABANNI$ agents and the *StartActiveBehavior* (START_STATE) for two other agent types. An agent will transit to the next state *ActiveBehavior* (ACTIVE_STATE), or in case of output agents *ActiveOutBehavior* (ACTIVE_STATE_OUT), if it receives input from the user for input neurons ($ABANNI_{DMAS}$), or from neurons of preceding levels for hidden and output neurons ($ABANNH_{DMAS}$ and $ABANNO_{DMAS}$). *ActiveBehavior* is the behavior in which a neuron calculates its state according to the received information, and information from agent's local database (containing information like neuron transfer function f). The only difference is that in the ACTIVE_STATE an agent generates ACL message *Change* and sends it to the connected neurons, while in ACTIVE_STATE_OUT it just calculates its state without generating ACL messages. Therefore AC-

TIVE_STATE is external behavior while ACTIVE_STATE_OUT is local agent behavior. On state completion, agents, representing input and hidden neurons, transit back to START_STATE if no error occurred. Output neurons transit to the CREATE_OUTPUT state (*CreateoutputBehavior*) that generates output information to the user. In case of an error each agent transits to DIE_STATE (*DieBehavior*) and generates an ACL message – *Die* that coordinates simultaneous termination of all agents. ACL message *Die* spreads among agents, since each agent sends it to all preceding and following agents.

In this section we have just provided the design of an ABANN agent, according to the proposed definition. This design will be tested as a part of future work.

§5 Conclusion

A distributed computation system can be based on a multi-agent system. Each computation component can be represented with an agent, and computationally distributed systems can benefit from agent technology approach. Agent properties (e.g. autonomy) allow adding individual characteristics to computationally distributed systems components. The formalization of the distributed computation multi-agent system agent as automata, provided in the article, facilitates agent design and realization.

A DCMAS agent is defined with states and transition events, and is behaving similar to a finite state machine. The main difference between a DCMAS agent and a finite state machine is that a DCMAS agent is taking into account a set of agent's internal behavior conditions that can cause state transition. Because an agent is defined with its behavior and the environment, the states and state transitions have to be incorporated into agent behavior and environment. And also, agent behavior and environment have to be incorporated into DCMAS agent states and state transitions. Since an agent behavior, representing state, can have internal conditions not caused by other states the introduction of the finite set of agent's internal behavior conditions was necessary for the DCMAS formalization.

The DCMAS development is facilitated if the layered architecture with nonsymbolic knowledge representation is chosen. The layered architecture enables the use of the evolutionary incremental development for the agent creation. The identified agent states can be added gradually to the agent what facilitates the development of the DCMAS agent.

The future research will be directed into investigating the benefits of introducing distributed computation multi-agent systems into other distributed computation techniques, models and methods. For fuzzy cognitive maps and artificial neural networks, the possibility to treat each network node and map concept as unique, to define different inference mechanisms and etc. can bring additional flexibility in modeling real-life systems.

References

1) Ferver, J., "Multi-agent Systems, An Introduction to Distributed Artificial Intelligence", Addison-Wesley, England, 1999
2) Wais, G. (Editor), "Multiagent Systems", MIT Press, Cambridge, Massachusetts, 1999
3) Wooldridge, M.; Jennings, N., "Intelligent Agents: Theory and Practice", *Knowledge Engineering Review* , 10(2), 115-152
4) Jennings, N.R.; "On agent-based software engineering", *Artificial Intelligence* 117, 277-296, 2000
5) Braubach L., Pokahr A., Bade D., Krempels K.H., Lamersdorf W.; "Deployment of Distributed Multi-agent Systems", *Fifth International Workshop Engineering Societies in the Agents*, p335–347, 2005
6) Brinn M., Berliner J., Helsinger A., Wright T., Dyson M., Rho S., Wells D.; "Extending the Limits of DMAS Survivability: The UltraLog Project," *IEEE Intelligent Systems 19 (5)*, p53-61, 2004
7) Gil Y., "On agents and grids: Creating the fabric for a new generation of distributed intelligent systems, " *Web Semantics: Science, Services and Agents on the World Wide Web 4 (2)*, p116-123, 2006
8) Ojha A.C., Pradhan S.K., Patra M.R., "Distributed Multi-agent System Architecture for Mobile Traders", *Proceedings on ICIT*, p214-216, 2007
9) Woodridge M. "An Introduction to MultiAgent Systems", *John Wiley & Sons,* New York, 2002
10) Zhong Z.; McCalley, J.D.; Vishwanathan, V.; Honavar, V., "Multiagent system solutions for distributed computing, communications, and data integration needs in the power industry," *IEEE Power Engineering Society General Meeting 1,* p45-49, 2004
11) Kosko B., "Fuzzy cognitive maps", *International Journal Man-Machine Studies 24*, p65-75, 1986
12) Kandasamy W.B.V., Smarandache F., "Fuzzy Cognitive Maps and Neutrosophic Cognitive Maps", *Books on Demand, ProQuest Information & Learning,*USA, 2003
13) Jennings N.R., Sycara K., Wooldridge M. "A Roadmap of Agent Research and Development", *Autonomous Agents and Multi-Agent Systems 1,*p 275-306, 1998
14) Russell S., Norvig P. "Artificial Intelligence: A Modern Approach", *Prentice Hall, Upper Saddle River,* USA, 2000

15) Gruber T.R., "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", *International Journal of Human Computer Studies* 43(5-6), p907-928, 1995

16) Jennings B., Brennan R., Gustavsson R., Feldt R., Pitt J., Prouskas K., Quantz J., "FIPA-compliant agents for real-time control of Intelligent Network traffic Computer Networks", 31 (19), p2017-2036, *Computer Networks* 31 (19), p2017-2036

17) Wang S., Xi L., Zhou B. "FBS-enhanced agent-based dynamic scheduling in FMS", *Engineering Applications of Artificial Intelligence*, Volume 21 (4), p644-657, 2008

18) Foundation for Intelligent Physical Agents (FIPA), *SC00061G FIPA ACL Message Structure Specification*, Geneva, Switzerland, 2002

19) Foundation for Intelligent Physical Agents (FIPA), *SC00001L FIPA Abstract Architecture Specification*, Geneva, Switzerland, 2002

20) Bellifemine F., Caire G., Rimassa, G., Poggi A. " JADE: A software framework for developing multi-agent applications. Lessons learned", *Information and Software Technology 50*, (12), p10-21, 2008

21) Bellifemine F., Rimassa, G., Poggi A., "JADE - A FIPA-Compliant Agent Framework", *Proc. of the 4th Int. Conf. and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, UK 1999

22) Stylios C.D., Groumpos P.P., "Fuzzy Cognitive Map in Modeling Supervisory Control Systems", *Journal of Intelligent & Fuzzy Systems*, 8(2), p83-98, 2000

23) Koulouriotis D.E., Diakoulakis I.E., Emiris D.M., Antonidakis E.N., Kaliakatsos I.A., " Efficiently modeling and controlling complex dynamic systems using evolutionary fuzzy cognitive maps", *International Journal of Computational Cognition*, 1 (2), p41-65, 2003.

24) Maja Stula, Darko Stipanicev, Ljiljana Bodrozic, "Intelligent Modeling with Agent-Based Fuzzy Cognitive Map", *International Journal of Intelligent Systems*, No. 25, p. 981-1004, 2010

25) Brooks R.A. "Intelligence without representation", *Artificial Intelligence,* 47, p139-159, 1991

26) Muller J.P., "The Design of Intelligent Agents: A Layered Approach", *Lecture Notes in Computer Science,* Springer, New York, USA, 1996

27) Pressman R.S., "Software Engineering: A Practitioner's Approach" McGraw-Hill, New York, USA, 2001

28) Bellifemine F., Caire G., Greenwod D. "Developing Multi-Agent Systems with JADE", John Wiley & Sons, New York, USA, 2007

29) McCulloch W.S., and Pitts, W. "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics, 5: 115-133, 1943.

30) Minsky M., and Papert, S. "Perceptrons", MIT Press, Cambridge, 1969.

31) Hecht-Nielsen R. "Neural Network Primer: Part I.", AI Expert, 4-51, 1989.

32) Haykin S. "Neural Networks: A Comprehensive Foundation", New York: Macmillan, USA, 1994.

33) Nigrin A., "Neural Networks for Pattern Recognition", MIT, A Bradford Book, USA,1993.

34) Kawato M., "Computational Schemes and Neural Network Models for Formation and Control of Multijoint Arm Trajectory", Neural Networks for Control, W.T. Miller, R.S. Sutton, and P.J. Werbos, Eds., Boston: MIT Press, 197-228., 1990.