Method of agents' state estimation in multiresolution multiagent simulation

D. PIERZCHAŁA, P. CZUBA

dariusz.pierzchala@wat.edu.pl, czuba.przemyslaw@wat.edu.pl

Military University of Technology, Faculty of Cybernetics
Institute of Computer and Information Systems
Urbanowicza Str. 2, 00-908 Warsaw, Poland

The paper proposes the multiagent techniques for approximation of agent's state in the multiresolution multiagent simulation. The key methods we have used for state aggregation and disaggregation are: consensus algorithm and formation control. The idea of the coordination of multiple agents has emerged from both observation and simulation of a collective behavior of biological entities. The consensus algorithms are commonly used for the cooperative control problems in the multiagent systems, whilst the formation control is the most popular and fundamental motion coordination problem in the multiagent systems, where agents converge to predefined geometric shapes.
The presented approach shows that multiagent methods seem to be very promising in multiresolution simulation. Consensus and formation control algorithms remove necessity to specify the much more complex algorithms for the aggregation and disaggregation needs.

Keywords: multiresolution multiagent simulation, multiagent systems, multiagent networks, formation control.

DOI: 10.5604/01.3001.0013.1460

1. Introduction

Multiagent Systems (MAS) is an interdisciplinary area of science which has been developing very dynamically in recent years. MAS is closely linked to the real world which inherently consists of a large number of constantly evolving and interacting components. Due to growing complexity of real world models, the use of classical methods in their analysis is often not satisfactory. This fact leads to convergence of that new field in computer science – multiagent system. It emerged from another field called *Distributed Artificial Intelligence* (DAI) and combines ideas from many disciplines, including: artificial intelligence (AI), sociology, economics and even philosophy [5].

The simplest definition of MAS might be as follows: "system composed of multiple interacting computing elements, known as *agents*'.

Agents are situated (Figure 1) in a specific environment and are capable of autonomous actions [6] (have decision making capabilities to create and adjust their actions in order to satisfy their intention objectives). Such entities are also capable of interacting with other agents what means that they can coordinate their actions with each other, communicate with one another and reach an agreement. That leads to the fundamental problem of multiagent systems which is a major part of the paper: agents coordination.

Every model is an abstraction of some part of reality. Nevertheless, its level of detail depends primarily on two factors [12]: scope (the extent of a system, input domain, and output range treated) and resolution (the level of detail at which the system components and their behaviors are depicted). Models also differ in terms of their representation (perspective).

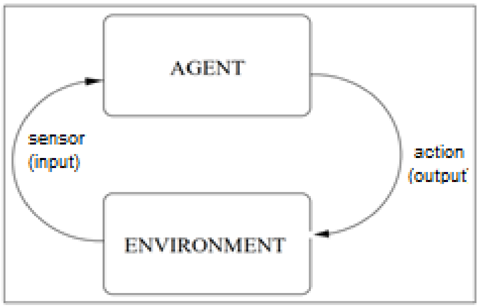

Fig. 1. An agent in its environment. Entity takes observations from the environment and according to them produces actions which affect it [7]

Multiresolution Modelling (MRM) [12] is a discipline which deals with solving the conceptual and representation differences that occur because of many resolution levels in the simulation of a simulation entity. A resolution might have many *dimensions.* A higher resolution can be perceived as more detailed representation of a modelled entity (a level of individual soldiers instead of a whole platoon). In MRM we typically say that a high resolution level corresponds to a low level of aggregation (high-resolution entity, HRE) and analogously – a low resolution level to a high level of aggregation (low-resolution entity, LRE), for instance in terms of entity's attributes.

When the two entities on different levels of resolution interact with each other, the problem with *consistency* might occur. There are many approaches for resolving the problem of transition between resolution levels [13]. In this paper we focus on the *cross-resolution method* (CRM) – *aggregation and disaggregation.*

Although the entities represent the same entity at an abstract level, they cannot interact correctly with each other while maintaining different levels of resolution. The typical solution for that problem is a dynamic change of entity resolution to hold the two entities on the same level of detail. *An aggregation* is a process which transits HRE to LRE (a state aggregation) and analogously a *disaggregation* amends LRE to HRE (a state disaggregation).

The other issues are complex adaptive systems (CAS) that are quite important motivation for MRM aggregated models. CAS exhibit coherent behaviors on macroscopic level which are not understandable in terms of microscopic rules that govern the system (*emergent behaviors*) [14]. It is said that intelligence is emergent behavior of biochemical systems. Although Schrödinger's paradox is now understood, we still do not know for instance how to explain consciousness in terms of proteins and DNA.

Multiresolution multiagent simulation brings together the above concepts. It simulates multiresolution models with use of multiagent methods.

The paper proposes to use multiagent techniques for estimation of agent's state in the multiresolution multiagent simulation. The key methods we have used for state aggregation and disaggregation are: consensus algorithm [15] and formation control [8]. They will be described in detail in later sections of this paper.

The paper is organized as follows. Section 2 gives a concise description of multiagent networks which are crucial in terms of the later parts. Section 3 contains a proposition for agent's state aggregation. In Section 4 authors describe the chosen disaggregation method. Section 5 contains description of a simulation package which has been used for the original implementation. Section 6 describes a case study with some experiments. And finally, the concluding remarks are given in Section 7.

2. Multiagent Networks

The *Multiagent Networks* [16] might be defined as a set of dynamic entities which cooperate via communication network to exchange messages. This fact allows them to coordinate their actions in order to fulfill collective goal having at the same time limited computing resources and communication as well as perception capabilities.

The common denominator of multiagent networks are the following properties:

- autonomic entities, potentially with decision making and communication capabilities between neighbors;
- existence of information exchange network.

The multiagent networks are being modelled with the use of Graph Theory methods and definitions (Figure 2).

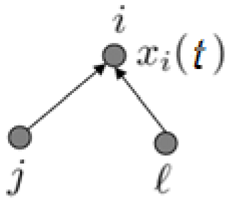

Fig. 2. Example of multiagent network modelled as a directed graph. Nodes *j* and *l* are predecessors of node *i* [16]

The following symbols are common multiagent network determinations:

- $x_i(t), i\epsilon N$ – state of node (agent) *i* at the time *t*;
- N_i – for undirected graph: set of neighbors of node *i*, for directed – set of predecessors;
- $I_i(t) = \{x_j(t) \mid j\epsilon\, N_i\}$ – information available to agent *i* at the time *t*;
- $x_i(t+1) = F_i(x_i(t), I_i(t)\,)$ – rule to determine next state of agent *i*.

The next sections contain description of the two most common use cases of multiagent networks: consensus and formation problems.

3. Aggregation method as the consensus algorithm

The consensus algorithms are commonly used for the cooperative control problems in the multiagent systems.

We can describe the consensus algorithm (protocol) as a set of interaction rules describing way of exchange of information between an agent and its neighbors. During “consensus process” the agent’s states are evolving over time. It is said that a system reached the consensus when values of states (that agents need to agree upon) are the same for every agent in a system.

The proposed method for aggregation of agent’s states is based on the following consensus protocol for a static in time topology and discrete time [17]:

$$x_i(t+1) = \frac{1}{\overline{\overline{N_i}}+1}\left(x_i(t) + \sum_{j\epsilon N_i} x_j(t)\right) \quad (1)$$

Agent’s state in the next time step is an arithmetic mean both of his and his neighbor’s states. If a topology was dynamic, a set of neighbors would be dependent on time [16].

The following limit [17] describes an expected result of the algorithm. It should be equal to the arithmetic mean of all N agents’ states:

$$\lim_{t\to\infty} x_i(t) = \frac{1}{\overline{\overline{N}}}\sum_{j\epsilon N} x_j(0), i\epsilon N \qquad (2)$$

Figure 3 shows the example evolution of agreeing by agents on the value that represents a state of lower resolution entity.

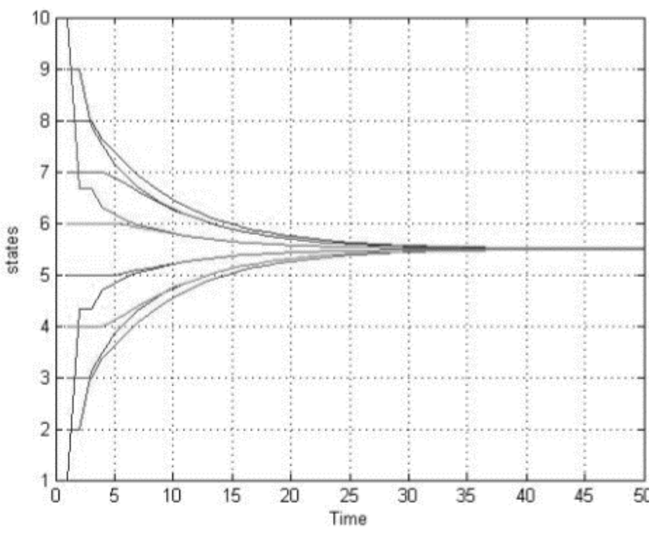

Fig. 3. Evolution of agent's states in time. In the end they share the same value [17]

4. Disaggregation method as the formation control

The idea of the coordination of multiple agents has emerged from both observation and simulation of a collective behavior of biological entities (like ants or birds). They achieve a complex group behavior through some network communication channels and an ordered motion coordination [8].

The formation control is the most popular and fundamental motion coordination problem in the multiagent systems, where agents converge to predefined geometric shapes. Agents need to obey local rules that are based on a partial knowledge about the position of its neighbors. The example of a multiagent task could be exploration, where a team of agents move according to specific formation in order to maximize their discovery skills.

The formation control strategies could be divided in to several categories. The first one is based on the *swarm intelligence* methods, where agents take actions accordingly to simple and reactive behavior rules. They keep a distance from the neighbors without maintaining a specific position in a formation and its shape. The example of such strategy is Reynold’s *Boid* model [9]. The second approach is based on specified communication network topology called *Formation Graph* (FG) [8]. The nodes represent agents positions whilst the edges possible communication channels between matched pairs of agents. A specified edge weight refers to a relative vector of the desired position between agents. Figure 4 shows example of FG.

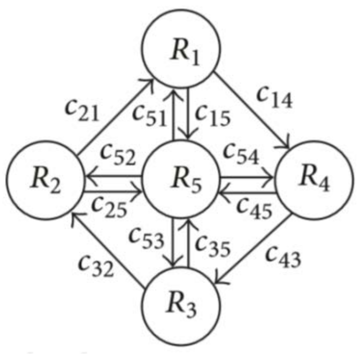

Fig. 4. Formation Graph example [8]

Most approaches to the formation control are analyzed from continuous-time point of view, for instance the complete FG, where exist a bidirectional communication between every pair of agents [10]. The discrete time formation control has been generally studied with the use of consensus algorithms which includes a relative position vectors [11]. This paper considers the agents converge to the static in time formation patterns and a discrete-time flow.

The formation control strategy used for a disaggregation is described precisely in [8]. It is based on Formation Graph. The starting position for every agent is equal to a state of LRE.

The desired relative position for every agent in a formation is described by the following formula:

$$z_i^*(t) = \frac{1}{\overline{\overline{N_i}}}\sum_{j\in N_i}(z_j + c_{ji}), i\epsilon N \qquad (3)$$

It is based on the desired relative position vectors (c_{ji}) between an agent i and his predecessors. The formation control strategy used for the disaggregation method is a formula:

$$u_i(t) = k(z_i(t) - z_i^*(t)), i = 1, \dots, n \qquad (4)$$

where k is an adjustment parameter.
The Figure 5 shows the simulation results for FG from Figure 4.

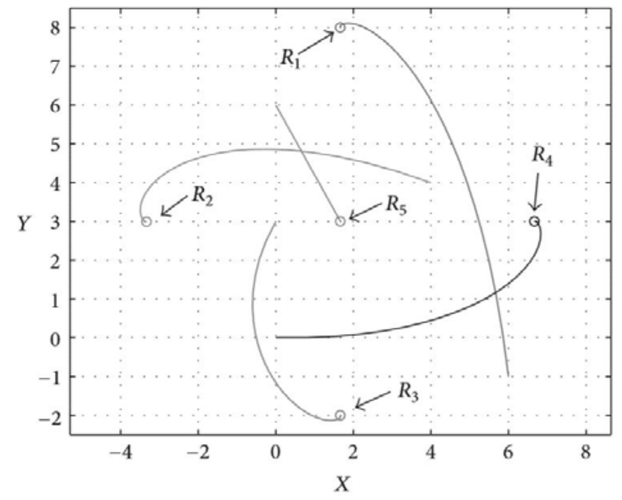

Fig. 5. Agent’s trajectories during calculation of desired formation positions [8]

5. Discrete simulation software package

Global trends in operation research indicate that computer simulation is one of the most common approaches to modelling cyberspace at the device, protocol, service and user levels. It increases the possibilities for quantitative and qualitative analysis and forecasting. It is a recognised alternative to experimenting with a real system or its physical prototype and to mathematical analysis. It replaces informal methods of reasoning based on experience and intuition. As a result of the development of computer systems, simulation has undergone the evolution of technologies and algorithms similar to typical IT systems: from a sequential simulation experiment performed on a stand-alone machine, through parallel, networked to distributed in virtual resources (so-called cloud computing). However, dedicated computer simulators, adequate but at the same time universal and open to modification, are essential for successful simulation. In our work, we emphasize these two features:

- openness to defining new algorithms for behaviours (activities) of objects;
- universality, i.e. the ability to adapt to various types of problems and levels of modelling.

We will define computer simulation as a quantitative and qualitative method of modelling in a formal language and then implementing in a computer program the features of systems (structural and behavioural) in order to experiment with the model and then study occurring processes during the simulation time [10]. The beginning of dynamic (in time meaning) computer simulation was the development of network models (Carl Adam Petri) and methods of system dynamics analysis (Jay W. Forrester) and then the languages and methods of continuous and discreet simulation: differential: (Lockheed – Digital Differential Analyzer Simulator DIDAS9), events (Harry Markowitz, Bernard Hausner – SIMSCRIPT), process interactions (Ole-Johan Dahl and Kristen Nygaard – Simula67, Knuth and McNeley – Algol SOL), selective activity (John Buxton and John Laski – Fortran CSL) and 3-phases (Keith Douglas Tocker – General Simulation Program GSP). The milestone was the language of Simula67 which, through its concept of class and object, moved away from the structural approach commonly used at the time and becoming the protoplast of objectivity. The response to the growing simulation demand are packages extending general purpose languages (e.g. C#, Java) with simulation services (dynamic calculation of state variables values, time control, generation of random numbers, dispersion of processes in the network). Thanks to these techniques, any evolution in language also implies new possibilities in simulation applications. Simulation experiments are carried out not only on individual computer stations, not only in local networks and grids, but also in services provided from cloud computing. In the case of the Java language, the following possibilities were used in the presented research: advanced collections of objects, functional interfaces and Lambda expressions, integration through open scripts with Groovy and Python languages.

The most popular and universal type of simulation is still being constructive simulation which is based on formal models of objects and their implementation in computer programs. The original Java-based discrete simulation software package (named DisSim) has been developed in order to facilitate creating constructive simulation software – both discrete and quasi-continuous as well as hybrid. DisSim is an object paradigm based package. The basic entity is an object '*o*' with its attributes:

- $O = \{o = <id, c>\}$, $c \in C^O$, $id \in N$ – a set of simulated objects of '*c*' class that are identified by an '*id*' with a unique value;
- C^O – a non-empty set of classes of modelled objects;
- A_c – a non-empty set of attributes specified for the $c \in C^O$ object class;
- $V_a^{\ c}$ – a set of acceptable values of the $a \in A_c$ attribute of the $c \in C^O$ objects class.

The C^O and A_c sets contain the numbers of modelled object classes and their attributes. At any time '*t*' of the simulation time each measurable feature of the system is represented by an ordered quadruple: $s_{o,a} = <o, a, v, t>$. The state of the modelled system will be the vector created by all attributes of all objects at the simulation time *t*: $S(t)=\{<o, a , v, t>\}$, $o \in O$. In the simulation $v \in V_a^{\ c}$ values of $a \in A_c$ attributes are determined as a result of the user-defined state change function. The concept of simulation time $t \in T$ is essential for a system model – it is a non-negative and non-decreasing real variable of value from $T \subseteq R^+ \cup \{0\}$. For the defined simulation moments t_i, t_j, t_k, for *T* as a collection of the moments when a system state changes there are applicable the following conditions:

- for a quasi-continuous passage of time: *T* is a subset of points in a certain range of non--negative real numbers such that $\forall t_i,t_k\ \exists t_j\ (t_i<t_j<t_k)$, so $|t_k-t_i| \le \varepsilon$, $\forall \varepsilon > 0$ – for any two moments there is a moment between them;
- for a discrete passage of time: *T* is a countable subset such that $\exists t_i,t_k\ \neg\exists t_j\ (t_i<t_j<t_k)$ – there are two such moments that there is no time between them.

Consequently, along with the assumptions for each object there are defined: the state vector with attributes and either the events or periodically executed equations that change the values of attributes (and finally a state of the system). The term event '*e*' from a finite set of events '*E*' is understood as the algorithmic change in object's state which is scheduled for a specific simulation moment *t*: $e_i = <t,\ f_e^S(t)>$, $t \in T$, $i=1...2^{ExT}$, where $f_e^S: SxT \rightarrow SxT$. According to this idea $S(t)$ is invariable over a period of time $[t_i, t_{i+1})$.

On the other side we have the Runge-Kutta numerical algorithm for the differential equations calculated upon the rule "Future state = Present state + change after time step". A sequence of chronologically (in a sense of simulation time) ordered change in state is the process of simulation. In general, it may be perceived as a multi-dimensional stochastic process where the individual elements of a state vector describe the various parameters of the system at the time *t*.

The current simulation time t^* is calculated either with the time stamp of the first event from the event calendar (a formula for an event-driven simulation) : $t^* = min\ \{t\!: e_i = <t,\ f_e^S(t)>\}$ or incrementally by a constant time step (a formula for step-by-step simulation) : $t^* = t^{i+1} = t^i + \Delta t$.

The result of an implementation in Java of the above model are the classes of DisSim package. The main classes of the package are presented in a simplified class diagram (Figure 6).

Every $o \in O$ simulation object of the $c \in C^O$ class inherits from the abstract *BasicSimEntity* class. Its A_c attributes store the values of the state vector. Each event $e = <t, f_e^S>$ is a state change created in objects inherited from the generic *BasicSimStateChange* class. An event is the result of a f_e^S state change function implemented as the *transition()* method of the *BasicSimStateChange* class. The event class attributes include: scheduled execution time and priority of the status change (*runSimTime*, *priority*) and an optional time step (*repetitionPeriod*) for either a discrete with a fixed step or a quasi- continuous simulation. For the latter type of simulation a special repeatable event class *RKEvent* is dedicated. After each time step (stored in *repetitionPeriod*), the values of equations describing the system state are determined in the method *transition()*. The exact form of the equations has to be determined in *Function<RKFunctParams, List<Double>>* by the DisSim user. The *Function* is a realization of Java functional interface thus an user is expected to implement lambda expressions and method references.

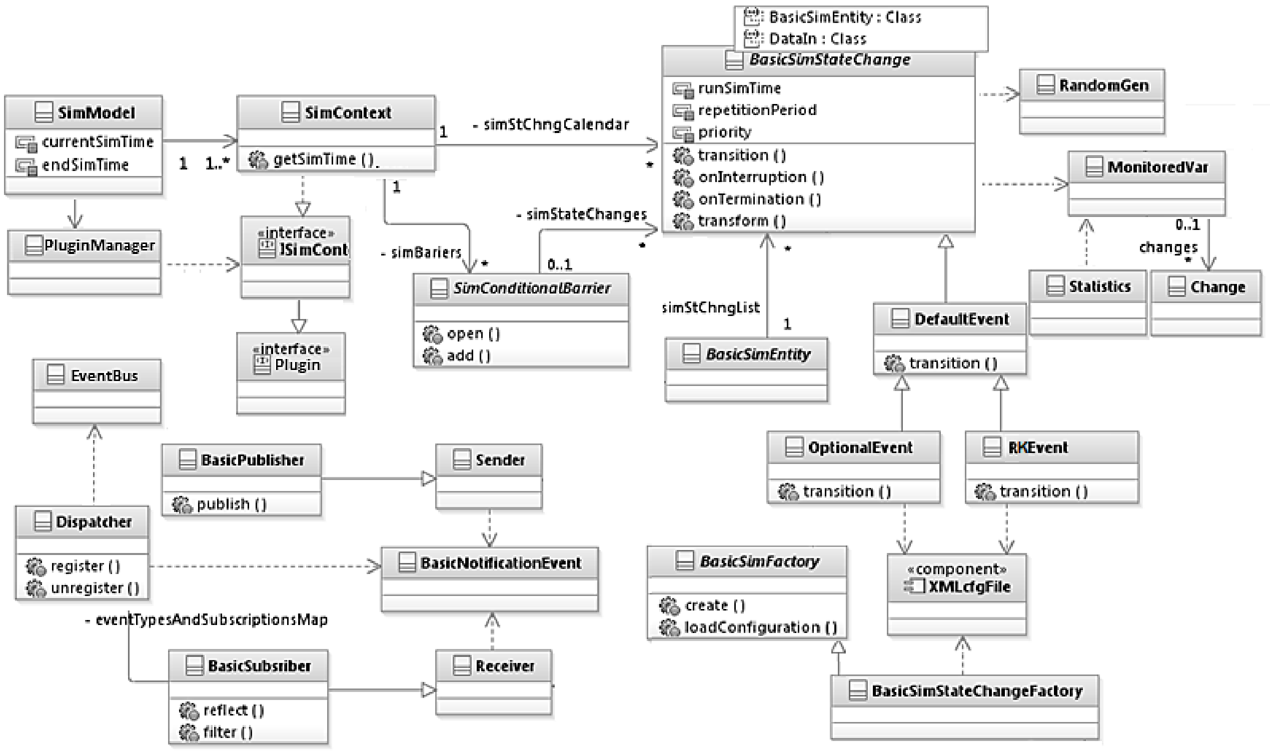

Fig. 6. Simplified class diagram of DisSim package

Moreover, DisSim is having classes responsible for message transmission via event bus, dynamic configuration of objects and open plug-in architecture. Additionally, as we indicate in previous section, an interpreter classes for Groovy scripts have been implemented. The integration between DisSim classes and Groovy scripts has been achieved on the base of *GroovyScriptEngine* and specialized methods to load/run scripts: *loadScriptByName(…)*, *getDeclaredMethod(method).invoke(…, …)*. For each object instance of *BasicSimStateChange* class a new algorithm of the method *transition()* might be defined in a separated Groovy script file.

Finally, codes from scripts will be loaded into the scripts' map in DisSim and launched by an interpreter during an execution of the *transition()* methods inside objects inherited from *BasicSimStateChange* and defined by the DisSim user.

6. Case study

Within scope of this paper the simulator of *Unmanned Aerial Vehicle* (UAV) teams has been implemented to properly test the presented ideas. It was developed with the use of DisSim package described in Section 5 and JavaFX library. UAV team adopts the specified formation according to its mission. Due to conservation of a computational power and the lack of necessity to observe individual units in every moment of the simulation, by default UAV teams are represented in an aggregated state (LRE). Figure 7 shows its graphical representation.

The course of simulation is following. At the beginning, the UAV teams are in the aggregated state and during the task of patrolling a specified terrain. When the team receives a signal about object that requires an action (searching team – a rescue, attacking team – an elimination), the LRE flies at the place of the event and performs the disaggregation. Figure 8 and Figure 9 shows the disaggregated formations for the search and attacking teams respectively. When the team finishes its mission, the entity aggregates its state and returns to patrolling.

The test experiments for originally implemented case study were made with use of *Java* method *System.currentTimeMillis()* which measures a duration of timeframe in milliseconds. We measure both aggregation and disaggregation timeframe for the different formations.

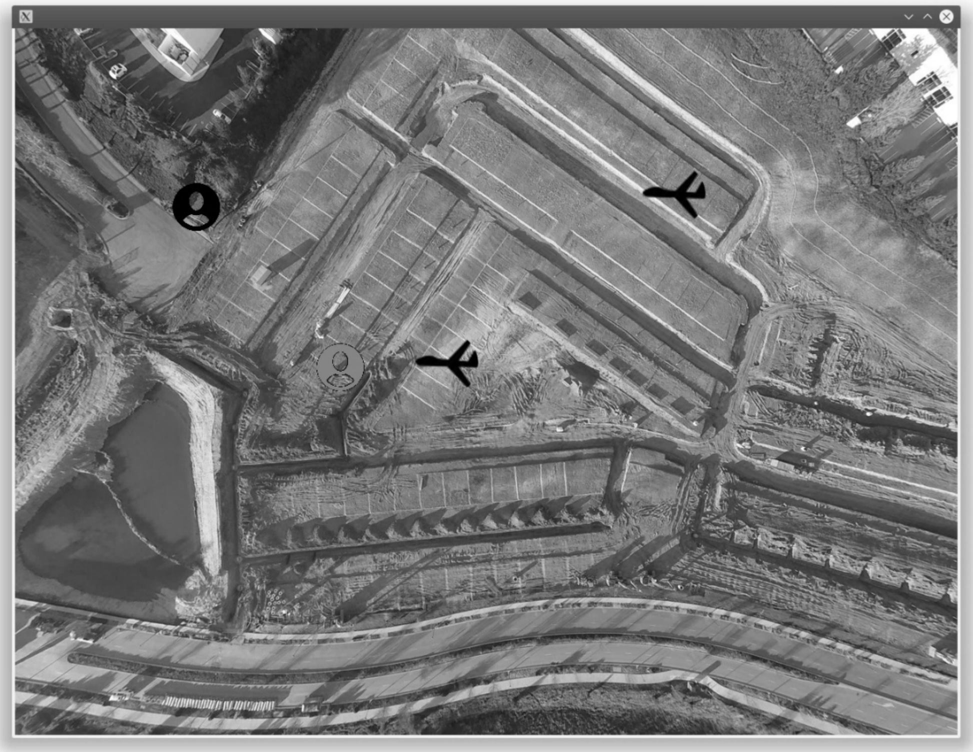
Fig. 6. UAV Simulator on aggregated level

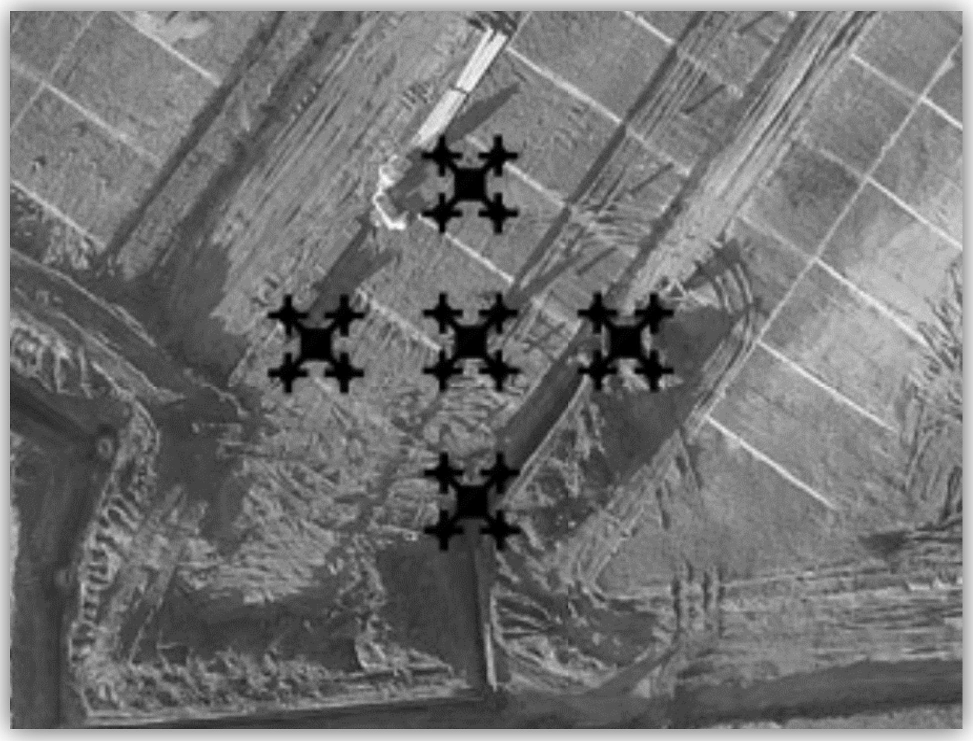
Fig. 7. Rescue formation with five nodes (agents)

For needs of the tests, it has been prepared additional formation – “Tortoise” (Figure 11). The results of the experiments are collectively presented in Table 1.

The network topology have huge significance in terms of disaggregation time. Attacking and searching formations have the same number of nodes, but different disaggregation times.

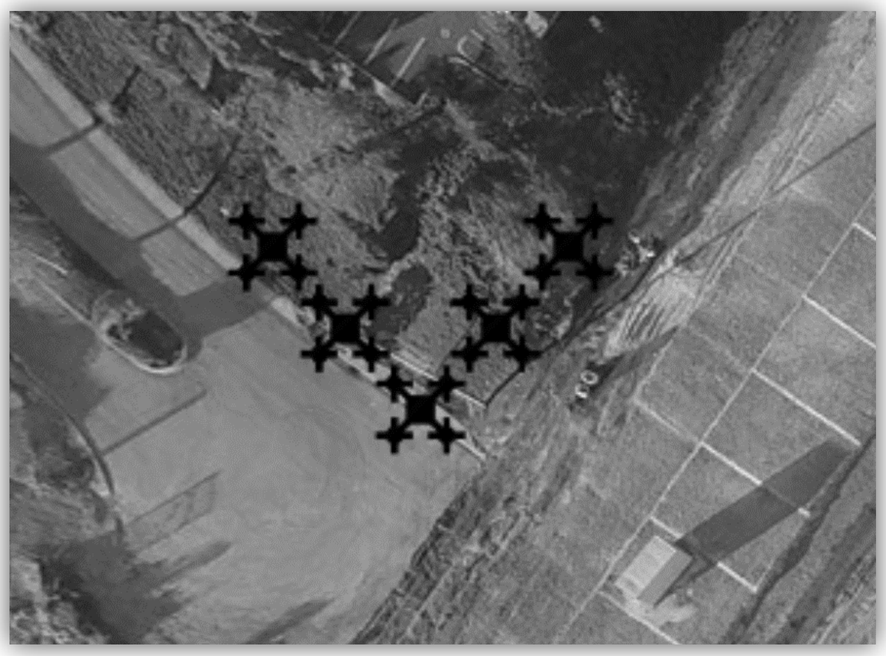

Fig. 8. Elimination formation with five nodes (agents)

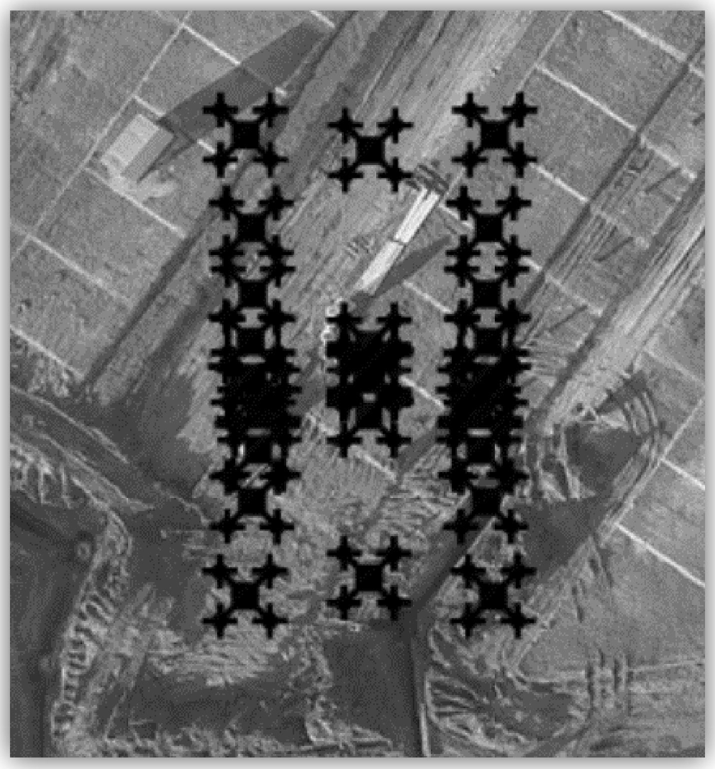

Fig. 9. "Tortoise" formation without "stabilization points"

The disaggregation effectiveness is dependent not only on a number of nodes in a network, but on the way how agents are connected in a network as well. The attacking formation has central agent which acts as "stabilization point". In this case, agents on the borders establish their positions relative to the neighbors, but at the same time they are blocking the UAVs in the middle of a formation which results in a long disaggregation time. Figure 10 shows "Tortoise" formation where agents are connected only to its closest neighbors.

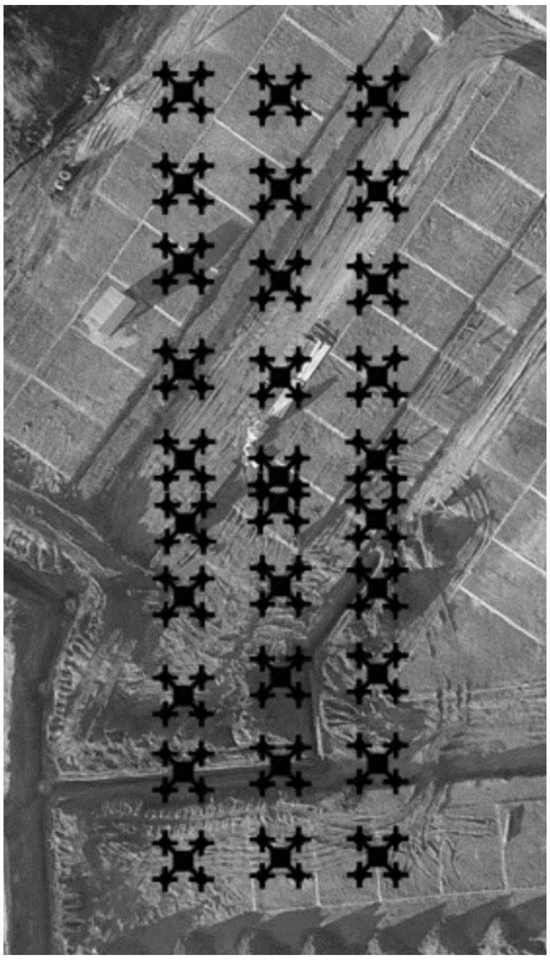

Fig. 10. “Tortoise” formation with fifteen nodes (agents)

Tab. 1. Results of experiments

Attacking formation										
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Aggregation [ms]	503	500	511	499	502	500	499	500	511	500
Disaggregation [ms]	6837	7101	7101	7111	7106	7102	7113	7112	7108	7106
Searching formation										
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Aggregation [ms]	1005	1000	699	700	701	700	699	701	699	700
Disaggregation [ms]	4825	4800	4801	4800	4799	4801	4801	4802	4800	4799
“Tortoise” formation										
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Aggregation [ms]	1528	1601	1599	1202	1202	1526	1601	1536	1599	1603
Disaggregation [ms]	32539	35524	44561	43630	35310	32616	35742	32732	35703	43869

By adding “stabilization points” between agents in the corners of formation, the acceleration of the process is achieved.

7. Conclusions

The presented approach shows that multiagent methods seem to be promising in multiresolution simulation. Consensus and Formation control algorithms, that work upon multiagent networks, remove necessity to specify the complex algorithms for the aggregation and disaggregation.

In the multiagent world it is hardly possible to predict and implement resolution change scenario for every possible situation. The multiagent methods allow to specify the general rules which give rise to complex behaviors.

The conception of multiresolution modelling and multiagent systems are inherently connected. The possibility to present an entity on aggregated level leads to saving computational power what is still being the problem in large multiagent simulations.

The Formation control is dependent on a Formation Graph specification. That implicates a situation of a programmer responsibility to ensure the effectiveness of a method. Moreover, the interesting idea would be to use machine learning methods for a formation control topology optimization.

Machine learning techniques could bring the aggregation and disaggregation scenarios specification to another state. Such algorithm might take into consideration variables, for instance, related to terrain and on this basis might choose optimal resolution level change method.

8. Bibliography

[1] Dyk M., Najgebauer A., Pierzchała D., “Agent-Based M&S of Smart Sensors for Knowledge Acquisition Inside the Internet of Things and Sensor Networks”, in: Intelligent Information and Database Systems, N.T. Nguyen et al. (Eds.), ACIIDS 2015, Part II, LNAI 9012, 224–236, Springer, 2015.

[2] Dyk M., Najgebauer A., Pierzchała D., “SenseSim: An Agent-Based and Discrete Event Simulator for Wireless Sensor Networks and the Internet of Things”, Proceedings IEEE World Forum on Internet of Things, WF-IoT 2015, 14–16 December, 2015, Milan, Italy, pp. 345–350, NY, USA, 2015.

[3] Dyk M., Najgebauer A., Pierzchała D., “Augmented perception using Internet of Things”, in: *Information Systems Architecture and Technology. Selected Aspects of Communication and Computational Systems*, A. Grzech, L. Borzemski, J. Świątek, Z. Wilimowska (Eds.), pp. 109–118, Oficyna Wydawnicza Politechniki Wrocławskiej, 2014.

[4] D. Pierzchała, “Application of Ontology and Rough Set Theory to Information Sharing in Multi-resolution Combat M&S”, in: *Advanced Approaches to Intelligent Information and Database Systems*, in series: Studies in Computational Intelligence, Vol. 551, pp. 193–203, Springer, 2014.

[5] Weiss G. (Ed.), *Multiagent Systems. A Modern Approach to Distributed Modern Approach to Artificial Intelligence*, The MIT Press, Massachusetts Institute of Technology, 1999.

[6] Wooldridge M., *An Introduction to Multiagent Systems*, Wiley Publishing, 2009.

[7] Wooldridge M., Jennings N.R., “Intelligent agents: theory and practice”, *The Knowledge Engineering Review*, Vol. 10, No.2, 115–152 (1995).

[8] Hernandez-Martinez E.G., Flores-Godoy J.J., Fernandez-Anaya G., “Decentralized Discrete-Time Formation Control for Multirobot Systems”, *Discrete Dynamics in Nature and Society*, Vol. 2013 (2013).

[9] Reynolds C.W., “Flocks, Herds, and Schools: A Distributed Behavioral Model”, in: SIGGRAPH ’87 Conference Proceedings, Vol. 21(4), pp. 25–34, ACM, New York, USA, 1987.

[10] Hernandez-Martinez E.G, Aranda--Bricaire E., “Non-collision conditions in multi-agent robots formation using local potential functions”, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA’08), pp. 3776–3781, Pasadena, California, USA, 2008.

[11] Ren W., Beard R. W., *Distributed Consensus in Multi-vehicle Cooperative Control*: *Theory and Applications*, in series: Communications and Control Engineering, Springer-Verlag, London 2008.

[12] Davis P.K., Bigelow J.H., *Experiments in Multiresolution Modeling* (*MRM*), RAND, 1998.

[13] Davis P. K., *An Introduction to Variable-Resolution Modeling and Cross-Resolution Model Connection*, RAND and the RAND Graduate School of Policy Studies, 1993.

[14] Holland J., *Hidden Order*: *How Adaptation Builds Complexity*, Addison-Wesley, Reading, Mass, 1995.
[15] Saber R. O. and Murray R. M., B "Consensus protocols for networks of dynamic agents", in: 2003 American Control Conference Proceedings, pp. 951–956, 2003.
[16] Mesbahi M., Egerstedt M., *Graph Theoretic Methods in Multiagent Networks*, Princeton University Press, 2010.
[17] Zhipu J., *Consensus Problem and Algorithms*, CDS 270-2: Lecture 8-1, May 2006.

Metoda wyznaczania stanu agentów w symulacji wieloagentowej o zmiennej rozdzielczości

D. PIERZCHAŁA, P. CZUBA

W artykule zaproponowano wieloagentowe podejście do wyznaczania stanu agenta w symulacji wielorozdzielczej (o zmiennej rozdzielczości) i wieloagentowej. Dwie kluczowe metody zastosowane do realizacji procesu agregacji i deagregacji stanów to: algorytm konsensusu i kontroli formacji. Idea koordynacji działań wielu agentów wyłoniła się z obserwacji oraz symulacji zbiorowych zachowań żywych istot. Algorytmy konsensusu są powszechnie stosowane w przypadku problemów sterowania kooperacyjnego w systemach wieloagentowych (konsensus oznacza osiągnięcie zgody na temat szczególnej wartości, która jest zależna od stanu wszystkich agentów w sieci). Natomiast kontrola formacji jest najpopularniejszym algorytmem w problemie koordynacji ruchu w systemach wielorobotowych, gdzie musi być spełniony warunek utrzymania predefiniowanego kształtu geometryczne formacji.
Przedstawione w artykule podejście pokazuje, że metody wielodyscyplinarne wydają się bardzo obiecujące w symulacji wielorozdzielczej. Algorytmy konsensu i kontroli formacji eliminują konieczność definiowania znacznie bardziej złożonych algorytmów na potrzeby agregacji i deagregacji.

Słowa kluczowe: symulacja wieloagentowa o zmiennej rozdzielczości, systemy wieloagentowe, sieci wieloagentowe, kontrola formacji grupy.