Task Hibernation in a Formal Model of Agent-Oriented Computing Systems

Maciej Smołka

Institute of Computer Science, Jagiellonian University, Kraków, Poland
`smolka@ii.uj.edu.pl`

Abstract. The paper contains recent enhancements of a formal model of agent-based computing systems. In such systems a computational task together with its data is enveloped in a shell to form a mobile agent. The shell carries the agent's logic, ie. abilities to make decisions about whether to migrate to a less loaded machine, split oneself or continue the task. The model describes an agent-based computational application as a controlled Markov chain. In this paper the operation of the agent hibernation, which is the last resort in the case of a server overload, is included in the model. This modification has an influence on the form of the state equations as well as the form of admissible control strategies.

1 Introduction

The multi-agent paradigm is already a classical design approach in a wide variety of domains (cf. [1], [2]), which can take advantage of the idea of mobile intelligent autonomous application unit. Computing systems are seldom considered as one of these domains. However, the concept of self-organising computational application composed of mobile tasks, which can move between interconnected computers according to a scheduling policy in order to find a better environment for executing themselves is well-known for several years. The article [3] describes such a system with a scheduling policy based on the heat conduction phenomenon.

A step forwards has been to put a task together with its data into an agent box, give the agent the ability to migrate, to communicate with other agents and to split itself (first of all its task) into two child agents (typically equal). In such a case a scheduling strategy may be incorporated in all agents' (distributed) intelligence. The strategy tells an agent in what order it should perform its activities (computations, migration, partitioning) to achieve its goals, which include typically the finishing of computations in the shortest possible time.

A multi-agent computational system of this type has been developed for the last several years (cf. [4]). It exploits a scheduling policy based on the phenomenon of the molecular diffusion in crystals (cf. [5], [6]). In this policy an agent makes decisions about actions to perform resting on its knowledge of the load of its computational node as well as the load of the node's direct neighbours. When an agent wants and is able to migrate, it chooses the least loaded neighbour as the target. The application of the multi-agent paradigm together with local, diffusion-based agent scheduling strategies provide us with a relatively simple decentralised management of large-scale distributed computations.

Many theoretical questions has been raised during the development of the above mentioned multi-agent system (MAS). The fundamental one is whether the used heuristic scheduling strategy is in any sense optimal or quasi-optimal. This in turn requires a precise definition of the optimality of a strategy. If the answer to the first question is negative, another problem is whether there exists an optimal policy at all and, if so, what are its characteristics.

To address these (and other) questions a formal model of computing multi-agent systems has been proposed (cf. [6–8]). The model is based on the stochastic optimal control theory. The model is still under development and its last version is described in [9]. It already provides us with a precise definition of the optimal task scheduling in our context as well as results on the existence of optimal scheduling strategies and optimality conditions. The above mentioned MAS has served as an example for the development of the model, but the latter has proven more general (and complicated).

In this paper the model is extended by considering the operation of the agent hibernation. Agents are hibernated eg. in an early stage of migration (cf. [4]), but migrations has been already considered in our model (cf. [9]). The interesting case of the hibernation occurs when a computational node is overloaded and an agent cannot find any neighbouring node to emigrate. The local MAS server will then serialize the agent to the disk and deserialize it when the load is sufficiently low. We modify our state equations to allow such hibernations.

Finally we present results on the existence and the characterisation of optimal scheduling strategies, which are adapted to the modified model.

2 System Architecture

First let us introduce to the reader the architectural principles of our exemplary computing MAS. They constitute the foundations and the starting point for the development of the mathematical model. Here we recall only the features most important for the model, for a more complete description and some implementation details we refer the reader to [4] and [5].

The suggested architecture of the system is composed of *a computational environment* (MAS platform) and *a computing application* being a collection of mobile agents called *Smart Solid Agents* (SSA). The computational environment is the triple $(\mathbf{N}, B_H, perf)$, where:

$\mathbf{N} = \{P_1, \ldots, P_N\}$, where P_i is a MAS server called *a Virtual Computational Node* (VCN). Each VCN can maintain a number of agents.

B_H is the connection topology $B_H = \{\mathcal{N}_1, \ldots, \mathcal{N}_N\}, \mathcal{N}_i \subset \mathbf{N}$ is a direct neighbourhood of P_i (including P_i as well).

$perf = \{perf_1, \ldots, perf_N\}, perf_i : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ is a family of functions where $perf_i$ describes the relative performance of VCN P_i with respect to the total memory request M^i_{total} of all agents allocated at the node.

The MAS platform is responsible for maintaining the basic functionalities of the computing agents. Namely it delivers the information about the local load concentration $\mathbf{L}_j$ and Q_j (see (2) and (3) below), it performs agent destruction, hibernation, partitioning and migration between neighbouring VCN's and finally it supports the transparent communication among agents.

We shall denote an SSA by A_i where index i stands for an unambiguous agent identifier (possibly a UUID). Each A_i contains its computational task and all data necessary for its computations. Every agent is also equipped with a shell which is responsible for the agent logic. At any time A_i is able to denominate the pair (E_i, M_i) where E_i is the estimated remaining computation time measured in units common for all agents of an application and M_i is the agent's RAM requirement in bytes. An agent may undertake autonomously one of the following actions:

– *continue* executing its internal task,
– *migrate* to a neighbouring VCN or
– decide to be *partitioned*, which results in creating two child agents $\{A_{i_j} = (T_{i_j}, S_{i_j})\}, j = 1, 2$.

We assume that in the case of the agent partitioning the following conditions hold:

$$E_i > E_{i_j}, \quad M_i > M_{i_j}, \quad j = 1, 2.$$

The parent SSA disappears after the partition.

A computing application may be characterised by the triple $(\mathbf{A}_t, G_t, Sch_t)$, $t \in [0, +\infty)$ where $\mathbf{A}_t$ is the set of application agents active at the time t, G_t is the tree representing the history of agents' partitioning until t. All agents active till t constitute the set of its nodes $\bigcup_{s=0}^{t} \mathbf{A}_t$, while the edges link parent agents to their children. All information on how to rebuild G_t is spread among all agents such that each of them knows only its neighbours in the tree. $\{Sch_t\}_{t \in [0, +\infty)}$ is the family of functions such that $Sch_t : \mathbf{A}_t \to \mathbf{N}$ is the current schedule of application agents among the MAS platform servers. The function is defined by the sets ω_j containing indices of agents allocated on each $P_j \in \mathbf{N}$. Every ω_j is locally stored and managed by P_j.

Each server $P_j \in \mathbf{N}$ asks periodically all local agents (allocated on P_j) for their requirements and computes the local load concentration

$$L_j = \frac{E^j_{total}}{perf_j(M^j_{total})} \text{ where } E^j_{total} = \sum_{i \in \omega_j} E_i \text{ and } M^j_{total} = \sum_{i \in \omega_j} M_i. \tag{1}$$

Then P_j communicates with neighbouring servers and establishes

$$\mathbf{L}_j = \left\{ (L_k, E^k_{total}, M^k_{total}, perf_k) : \ P_k \in N_j \right\} \tag{2}$$

as well as the set of node indices

$$Q_j = \{ k \neq j : \ P_k \in N_j, \ L_j - L_k > 0 \}. \tag{3}$$

3 Global State of a Computing Application

In this section we introduce some key concepts appearing in our formal model of computing multi-agent systems. The detailed description of the model may be found in [9].

The key idea behind the model is to abandon the considering of single agents' behaviour in favour of observing a global quantity characterising the state of a computational application in an appropriate way. To this end we have introduced the notion of the *vector weight of an agent*, which is the mapping

$$w : \mathbb{N} \times \mathbf{A} \longrightarrow \mathbb{R}_{+}^{M}$$

with $M \geq 1$. The weight has at least one *positive* component as long as the agent is *active* (ie. its task is being executed) or *hibernated.* In other words the equality

$$w_t(A_i) = 0 \tag{4}$$

means that the agent A_i does not exist yet or already. Note that we observe the application state in discrete time moments, so the set of times is $\mathbb{N}$. We assume that the dependency of the total weight of child agents after partition upon their parent's weight before partition is well-known and linear, i.e. there is a matrix $\mathbf{P} \in \mathbb{R}_{+}^{M\times M}$ such that in the case of partition $A \rightarrow \{A_1, A_2\}$ we have

$$w_{t+1}(A_1) + w_{t+1}(A_2) = \mathbf{P} w_t(A). \tag{5}$$

The single agent weight is only an auxiliary notion needed to define what shall be one of the main observed global quantities, ie. the *total weight of all agents allocated on a virtual node P at any time t*, which is

$$W_t(P) = \sum_{Sch_t(A)=P} w_t(A).$$

In previous papers W_t was the system state, but as we want to differentiate between active and hibernated agents, we shall split the total weight into two terms

$$W_t(P) = W_t^a(P) + W_t^h(P),$$

where $W_t^a(P)$ is the total weight of *active* agents and $W_t^h(P)$ is the total weight of *hibernated* agents. We shall observe the evolution of both quantities separately.

If the components of w include E_i and M_i defined in Sec. 2 (as in [7], [8]), then:

- $M_i > 0$ for active agents and $M_i = 0$ for hibernated ones,
- E_i is positive till the agent destruction and does not change as long as the agent is hibernated.

In this case obviously E_{total}^i and M_{total}^i will be among the components of W. But in general it may be convenient to find other *state variables* (see [9] for considerations on that topic). In any case both E^i_{total} and M^i_{total} should remain *observables* of our system.

In the sequel we shall assume that the number of virtual nodes

$$\sharp\mathbf{N} = N$$

is *fixed*. We could also assume that it is *bounded*, but this is not a big generalisation. For the sake of conciseness we introduce the notation

$$W^{a,j}_t = W^a_t(P_j), \quad W^{h,j}_t = W^h_t(P_j)$$

for $j = 1, \ldots, N$. Then W^a_t and W^h_t may be interpreted as vectors from $\mathbb{R}^{MN}_+$ or, if it is more convenient, as nonnegative $M \times N$ matrices.

According to the interpretation of (4) the equality $W_t = 0$, which is equivalent to $W^a_t = W^h_t = 0$, means that at the time t there are neither active nor hibernated agents, ie. the computations are finished. In other words 0 is the final state of the application's evolution.

4 Global State Evolution

In this section we shall formulate the equations of the evolution of our state variables, ie. W^a_t and W^h_t. They are expected to be a generalisation of the state equations presented in [9], so they should reduce to those equations in the absence of hibernations. Consequently they shall be stochastic difference equations, which means that the pair (W^a_t, W^h_t) shall form a discrete stochastic process.

First of all let us recall what has been called the 'established' evolution equation. It has been the equation showing the evolution of an application in the absence of agent migrations and partitions and it has the following form.

$$W_{t+1} = F(W_t, \xi_t) \tag{6}$$

where F is a given mapping and $(\xi_t)_{t=0,1,\ldots}$ is a given sequence of random variables representing the background load influence. We assume that ξ_t are mutually independent, identically distributed and have a common finite set of values Ξ, which is justified in many natural situations (cf. [8], this paper also considers more general assumptions on the background load).

In our case, we extend the meaning of the 'established' evolution by excluding hibernations as well. This results in the following conditions

$$W^h_t = 0, \quad W^a_t = W_t.$$

Thus we can rewrite (6) to obtain

$$\begin{cases} W^a_{t+1} = F(W^a_t, \xi_t) \\ W^h_{t+1} = 0. \end{cases} \tag{7}$$

Since 0 has to be *an absorbing state* of $\overline{W}_t = (W_t^a, W_t^h)$, we need to assume that for every t

$$F(0, \xi_t) = 0 \tag{8}$$

with probability 1. To guarantee reaching the final state we need also another assumption stating that there exists $t > 0$ such that for every initial condition $\widehat{W}$ and W_t evolving according to (6) we have

$$\Pr\left(W_t = 0 \mid W_0 = \widehat{W}\right) > 0. \tag{9}$$

It is easy to see that a similar condition holds for $\overline{W}_t$ and the natural initial state $(\widehat{W}, 0)$. A desired consequence of (9) (cf. [10]) is that with any initial condition our application will eventually finish the computations, which makes the assumption quite useful.

The equations of migration and partition are almost the same as in [9], the only difference is that they describe the behaviour of W_t^a (W_t^h does not change during a migration or a partition).

In the case of sole hibernations and dehibernations at node P_j (and no migrations, partitions or 'established' evolution) the system shall behave according to the following equation.

$$\begin{cases} W_{t+1}^{a,j} &= \left(I - \mathrm{diag}(u_t^{h,j}(W_t^a))\right) W_t^{a,j} + \mathrm{diag}(u_t^{a,j}(\overline{W}_t) W_t^{h,j} \\ W_{t+1}^{h,j} &= \left(I - \mathrm{diag}(u_t^{a,j}(\overline{W}_t))\right) W_t^{h,j} + \mathrm{diag}(u_t^{h,j}(W_t^a)) W_t^{a,j} \\ W_{t+1}^{a,i} &= W_t^{a,i} \\ W_{t+1}^{h,i} &= W_t^{h,i} \qquad \text{for } i \neq j. \end{cases} \tag{10}$$

$u_t^{h,j} : \mathbb{R}_+^{MN} \to [0,1]^M$ and $u_t^{a,j} : \mathbb{R}_+^{MN} \times \mathbb{R}_+^{MN} \to [0,1]^M$ are the proportions of components of the total weights of agents, respectively, hibernated and activated (dehibernated) to the corresponding proportions of the total weights of all agents allocated at node P_j at the moment t. We assume that the decision of hibernating some agents is the result of a resource shortage. As hibernated agents do not make use of any resources interesting from our point of view (we assume that server disks are large enough) this decision is based on the weight of active agents only. In other words u_t^h does not depend on W_t^h. In contrast the decision of reactivating some hibernated agents may depend on some of their features (eg. even if some RAM is free, every single hibernated agent may be too big to be activated), so u_t^a depends on both weight components.

Now we are in position to present the complete state equations. They are an extension of the state equations presented in [9] and they are constructed in a similar way, ie. as a combination of the above mentioned simplified equations reducing to these equations in their described particular context. We propose the following combination.

$$\begin{cases} W_{t+1}^{a,i} & = g^i\Big(F^i(\widetilde{W}_t^a,\xi_t),\ \mathbf{P}\,\mathrm{diag}(u_t^{ii}(W_t^a))\,W_t^{a,i},\ \sum_{j\neq i}\mathrm{diag}(u_t^{ji}(W_t^a))\,W_t^{a,j}, \\ & \qquad \mathrm{diag}(u_t^{a,i}(\overline{W}_t))\,W_t^{h,i}\Big) \\ W_{t+1}^{h,i} & = (I-\mathrm{diag}(u_t^{a,i}(\overline{W}_t)))\,W_t^{h,i} + \mathrm{diag}(u_t^{h,i}(W_t^a))\,W_t^{a,i} \\ W_0^{a,i} & = \widehat{W}^i \\ W_0^{h,i} & = 0 \end{cases} \tag{11}$$

for $i = 1,\dots,N$, where

$$\widetilde{W}_t^{a,i} = \Big(I - \textstyle\sum_{k=1}^N \mathrm{diag}(u_t^{ik}(W_t^a)) - \mathrm{diag}(u_t^{h,i}(W_t^a))\Big)\,W_t^{a,i}$$

and $\widehat{W}$ is a given initial state. For (11) to reduce to simplified equations we need an assumption on g, which may be eg. $g(s,0,0,0)=s$, $g(0,p,0,0)=p$, $g(0,0,m,0)=m$, $g(0,0,0,h)=h$. Note that in [8] and earlier papers we used a stronger condition, ie. $g(s,p,m,h)=s+p+m+h$.

It follows that W_t is a *controlled stochastic process* with a *control strategy*

$$\pi = (\mathbf{u}_t)_{t\in\mathbb{N}}, \quad \mathbf{u}_t = (u_t, u_t^a, u_t^h) : \mathbb{R}_+^{MN} \longrightarrow U. \tag{12}$$

The control set U contains such elements $\mathbf{a} = (\alpha,\alpha^a,\alpha^h)$ from $[0,1]^{M(N\times N)} \times [0,1]^{MN}\times[0,1]^{MN}$ that satisfy at least the following conditions for $m=1,\dots,M$.

$$\alpha_m^{ij}\cdot\alpha_m^{ji} = 0 \text{ for } i\neq j, \quad \alpha_m^{i1}+\cdots+\alpha_m^{iN}+\alpha_m^{h,i}\leq 1 \text{ for } i=1,\dots,N. \tag{13}$$

The first equation in (13) can be interpreted in the following way: *at a given time migrations between two nodes may happen in only one direction.* The second equality means that *the number of agents leaving a node, partitioned or hibernated at the node must not exceed the number of agents active at the node.*

Remark 1. It is easy to see that the control set U defined by the conditions (13) is compact (and so are of course its closed subsets).

As in [9] we do not take the whole $\mathbb{R}_+^{MN}\times\mathbb{R}_+^{MN}$ as the state space. Instead we choose finite subsets S^a, S^h of $\mathbb{N}^{MN}$ both containing 0 and $S = S^a\times S^h = \{s_0=(0,0), s_1,\dots,s_K\}$ shall be the state space. Consequently we assume that F and g have values in S^a. Additionally, we need also to assume that for every $t\in\mathbb{N}$ and $\overline{W}=(W^a,W^h)\in S$

$$u_t(\overline{W}) \in U_{\overline{W}} = \big\{\mathbf{a}\in U:\ G^a(\overline{W},\mathbf{a},\xi)\in S^a,\ G^h(\overline{W},\mathbf{a})\in S^h \text{ for } \xi\in\varXi\big\}$$

where G^a and G^h denote the right hand sides of, respectively, the first and the second equation in (11). The above equality implies that $G^a(\overline{W},0,\xi) = F(W^a,\xi)\in S^a$ and $G^h(\overline{W},0) = W^h\in S^h$ for any $\overline{W}$ and ξ, which means that $(0,0)\in U_{\overline{W}}$, therefore $U_{\overline{W}}$ is nonempty for every $\overline{W}\in S$. On the other hand we have $G^a(0,\mathbf{a},\xi)=F(0,\xi)=0$ and $G^h(0,\mathbf{a})=0$, i.e. 0 is an absorbing state of $\overline{W}_t$ independently of a chosen control strategy.

Remark 2. Similarly to [9] it remains true that $\overline{W}_t$ is a *controlled Markov chain* with transition probabilities $p_{ij}(\mathbf{a}) = \Pr(G(s_i, \mathbf{a}, \xi_0) = s_j)$ for $i, j = 0, \dots, K$, $\mathbf{a} \in U_{s_i}$, $G = (G^a, G^h)$. The transition matrix for the control $\mathbf{u}$ is $P(\mathbf{u}) = [p_{ij}(\mathbf{u}(s_i))]_{i,j=0,\dots,K}$.

5 Optimal Scheduling Problem

Let us now recall (after [7]) the definition of the optimal scheduling for a computing MAS in terms of the stochastic optimal control theory. We have already the state equations (11), so we need also a cost functional and a set of admissible controls.

The general form of considered cost functionals is

$$V(\pi; s) = E[\textstyle\sum_{t=0}^{\infty} k(\overline{W}_t, \mathbf{u}_t(\overline{W}_t))] \tag{14}$$

where π is a control strategy (12) and $s = (s^a, 0)$ is the initial state of $\overline{W}_t$, i.e. $\overline{W}_0 = s$. Since 0 is an absorbing state we shall always assume that remaining at 0 has no cost, i.e. $k(0, \cdot) = 0$. This condition guarantees that the overall cost can be finite.

The form of the set of admissible strategies is a modification of the one used in [9], namely

$$\mathbf{U} = \bigl\{\pi : \ \mathbf{u}_t(\overline{W}) \in U_{\overline{W}}, \ t \in \mathbb{N}\bigr\}.$$

Now we can formulate the *optimal scheduling problem.* Namely given an initial configuration $(\widehat{W}, 0)$ we look for a control strategy $\pi^* \in \mathbf{U}$ such that

$$V(\pi^*; (\widehat{W}, 0)) = \min\bigl\{V(\pi; (\widehat{W}, 0)) : \ \pi \in \mathbf{U}, \ \overline{W}_t \text{ is a solution of (11)}\bigr\}. \tag{15}$$

In other words an *optimal scheduling* for W_t is a control strategy π^* realising the minimum in (15).

Our main general tool for proving the existence of optimal scheduling strategies is [9, Prop. 4]. Its key assumptions are (R1) and (R2). They are expressed in terms of special properties of the transition matrix $P(\mathbf{u})$, but they mean that K-step (for (R2) n-step for some n) probability of reaching 0 from *every* initial state is positive provided we use:

- for (R1) any stationary strategy $\pi = (\mathbf{u}, \mathbf{u}, \dots)$;
- for (R2) one particular stationary strategy.

In (R2) we have to impose a stronger assumption on the one-step cost k, nevertheless it is much easier to check than (R1). In our case thanks to (9) the assumption (R2) is satisfied eg. for the zero control strategy $\pi = (0, 0, \dots)$.

For this reason in the sequel we shall consider only costs which satisfy the second part of (R2), ie.

$$k(s, \mathbf{a}) \geq \varepsilon > 0, \quad \text{for } s \neq 0, \ \mathbf{a} \in U_s. \tag{16}$$

Among cost functionals presented in [9] only the expected total time of computations V_T satisfies automatically (16). Let us recall that it has the following form.

$$V_T(\pi; s) = E\big[\inf\{t \geq 0 : W_t = 0\} - 1\big]. \tag{17}$$

Remaining two functionals (V_L and V_M) do not satisfy the assumption (16), but we can reduce the problem by adding a term cV_T with (maybe small) $c > 0$ to both of them.

First of the two (the one promoting good load balancing) after such a modification has the following form.

$$V_{LT}(\pi; s) = E\left[\sum_{t=0}^{\infty} \sum_{i=1}^{N} (L_t^i - \overline{L}_t)^2\right] + cV_T(\pi; s), \quad \overline{L}_t = \tfrac{1}{N} \sum_{i=1}^{N} L_t^i \tag{18}$$

where L_t^i is the load concentration (1) at P_i at the moment t. These quantities are well defined because we have assumed that E_{total}^i and M_{total}^i are observables of our system.

The last cost functional from [9] (penalising migrations) after the modification has the following form.

$$V_{MT}(\pi; s) = E\left[\sum_{t=0}^{\infty} \sum_{m=1}^{M} \sum_{i \neq j} \mu_m^{ij}(u_{m,t}^{ij}(W_t^a))\right] + cV_T(\pi; s). \tag{19}$$

$\mu_m^{ij} : [0, 1] \to \mathbb{R}_+$ allows us to take into account the *distance* between P_i and P_j.

Considerations accompanying (15) along with [9, Prop. 4] result in the following corollary.

Corollary 3. *Problem* (15) *for V_T, V_{LT} or V_{MT} has the unique solution.*

Finally, let us recall the optimality conditions presented in [9, Prop. 6] and rewrite it in our context in the following corollary.

Corollary 4. *Let V denote any of the functionals V_T, V_{LT}, V_{MT}. The optimal solution of* (15) *is a stationary strategy $\pi^* = \mathbf{u}^\infty = (\mathbf{u}, \mathbf{u}, \dots)$ and it is the unique solution of the equation*

$$V(\pi^*; s) = \min_{\mathbf{a} \in U_s} \left[\sum_{j=1}^{K} p_{ij}(\mathbf{a}) V(\pi^*; s_j) + k(s, \mathbf{a})\right]. \tag{20}$$

The solution of (20) *exists and it is the optimal solution of* (15).

(20) can be solved by means of an iterative procedure such as Gauss-Seidel. The complexity of the computational problem depends first of all on the size of the state space S, which in general is expected to be quite big. To make things better in a typical situation many states are inaccessible from one another so the matrix p_{ij} is sparse. The complexity is increased on the other hand by the minimisation and depends on the size (and the structure) of control sets U_s.

6 Conclusions

The presented MAS architecture along with diffusion-based agent scheduling strategies form a relatively easy to manage and quite efficient framework for large-scale distributed computations. The presented mathematical model provides us with a precise definition of optimal task scheduling in such an environment. It also gives us some useful results concerning the existence of optimal scheduling strategies (Cor. 3) as well as the optimality conditions (Cor. 4). The latter show that the choice of scheduling strategies utilised during tests (cf. [5]) was proper, because it is a stationary strategy and according to Cor. 4 the optimal strategy belongs to that class. In this paper the formal model has been extended to consider agent hibernations neglected so far. Also the existence results and the optimality conditions has been adapted appropriately. It appears that it is an important extension and omitting hibernations in some cases might result in a wrong observation of the state of a computing application. Further plans concerning the model include detailed studies on the form of the optimal strategies and, on the other hand, finishing some experiments (and starting some new ones) expected to extend the model's empirical basis.

References

1. Bradshaw, J.M., ed.: Software Agents. AAAI Press (1997)
2. Wooldridge, M.: An Introduction to Multi-agent Systems. Wiley (2002)
3. Luque, E., Ripoll, A., Cortés, A., Margalef, T.: A distributed diffusion method for dynamic load balancing on parallel computers. In: Proceedings of EUROMICRO Workshop on Parallel and Distributed Processing, San Remo, Italy, IEEE Computer Society Press (1995) 43–50
4. Uhruski, P., Grochowski, M., Schaefer, R.: Multi-agent computing system in a heterogeneous network. In: Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002), Warsaw, Poland, IEEE Computer Society Press (2002) 233–238
5. Grochowski, M., Schaefer, R., Uhruski, P.: Diffusion based scheduling in the agent-oriented computing systems. Lecture Notes in Computer Science **3019** (2004) 97–104
6. Grochowski, M., Smołka, M., Schaefer, R.: Architectural principles and scheduling strategies for computing agent systems. Fundamenta Informaticae **71**(1) (2006) 15–26
7. Smołka, M., Grochowski, M., Uhruski, P., Schaefer, R.: The dynamics of computing agent systems. Lecture Notes in Computer Science **3516** (2005) 727–734
8. Smołka, M., Schaefer, R.: Computing MAS dynamics considering the background load. Lecture Notes in Computer Science **3993** (2006) 799–806
9. Smołka, M.: A formal model of multi-agent computations. Lecture Notes in Computer Science (2008) to appear.
10. Kushner, H.: Introduction to Stochastic Control. Holt, Rinehart and Winston (1971)