Decentralized Approach for Autonomous Mechatronic Oil Extraction

Igor Kalyaev, Anatoly Kalyaev and Iakov Korovin
Southern Federal University, Taganrog, Russia

***Abstract*—In the paper we tried to depict the process of creation of autonomous mechatronic oil extraction production (AMOEP). Basically we propose that in its membership AMOEP includes a variety of well agents (WA) of different purposes to be proceeded. The key point of such AMOEPs must be a controller, which functions form the construction time chart of manufacture of the product from the WA, included in the AMOEP, as well as a plan for transport operations. Thus we propose new decentralized method of scheduling AMOEPs, based on negotiations of independent agents, implemented on different wells. To realize that, we consider to represent tasks for oil extraction in the form of an acyclic graph, each vertex is assigned a certain operation, performed by one of the WA. The distribution of tasks between the WAs is performed by their collective interaction. Also a detailed algorithm of the software well agent is presented.**

Keywords—autonomous mechatronic oil extraction process; decentralized dispatcher; task graph, software agents; collective decision making

I. Introduction

Nowadays we can see, how technological progress raises importance of high-tech products in oil industry worldwide implementation and leads to a problem of creating autonomous mechatronic oil extraction production (AMOEP), capable to quickly fulfil the industrial tasks, according to investment strategy of the company. This AMOEP should be composed of a wide range of well agents (WA) of different applications. The customer creates and sends via the cloud the task to extract discrete volume of hydrocarbon, using the WA, included in the AMOEP. According to this automatic task a manager must build AMOEPs plan (schedule time) of fulfilling the product. Next, the plan is launched in the production, according to the investment strategy of the enterprise.

The organization of such AMOEPs requires the solution of two fundamental problems: firstly, the need to develop a "form" representation of the job, "understandable" for the dispatcher of AMOEPs, and secondly, to develop a method for automatic generation of plan (timetable) of performing tasks, using the WAs. The solution of these problems is presented as follows.

II. The Formal Statement of the Problem

We assume that the AMOEP includes a set of WA $R_1, R_2, ...R_N$ so as as two warehouses - warehouse of components and a warehouse of extracted production. All WA and stores share a common transport line, by means of which components, articles and the workpiece can be transferred between the warehouses and WA.

Assuming that each WA R_i can perform a set of operations $\mathbf{A}_i =< A_1^i, A_2^i, ...A_L^i >$ $(i = 1, 2, ..., N)$, and generally $\mathbf{A}_i \neq \mathbf{A}_j$ $(j = 1, 2, ..., i-1, i+1, ..., N)$. We assume that the WA R_i performs A_l^i $(l = 1, 2, ..., L)$ during $t_i(A_l^i)$, and runtime operations identical various WA same. Also assume that the transport time of components and workpieces between the individual WA, and also between the WA and warehouses is $t_{\Pi}(S)$ where S - transport line length.

We assume that the AMOEP at random times over the corporative net (cloud) enters a plurality (stream) of different specifications for the manufacture of articles $\mathbf{Z} =< Z_1, Z_2, ..., Z_M >$, wherein for each job Z_l operator sets point in time $T_{\max}^l$ to which he wishes to achieve the oil extracted.

The aim of the AMOEP is to perform all tasks in the manufacturing of products $\mathbf{Z} =< Z_1, Z_2, ..., Z_M >$ to established operators in time.

Formalization of jobs. In order to AMOEP could carry out the production of some products $Z_l \in \mathbf{Z}$, the job for its production must be submitted by the customer in a standardized formalized form, understandable for automatic AMOEP controller. The task $Z_l \in \mathbf{Z}$ can be represented as an acyclic graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$, each node $q_j \in \mathbf{Q}_l$ which is attributed to a certain operation A_j, belonging to the set of operations, performed by some WA, and if two vertices q_j and q_{j+1} are connected by arc $x(q_j, q_{j+1})$ it means that the operation A_{j+1} attributed to the top q_{j+1} must be performed to complete the operation A_j attributed to the top q_j. Input vertex $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ defines operations for the delivery of parts, needed for the manufacture of products from a warehouse and a final vertex defines an operation to place at the warehouse of the final product, obtained as a result of all of its manufacturing program.

III. Principles of the AMOEP Manager Organization

After the job Z_l in the manufacture of products is formalized in the form of a graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$, it is received the AMOEP controller whose functions consist in the construction plan (time chart) articles of manufacture, i.e., in the distribution of individual operations jobs Z_l bound between them to perform a specified time in accordance with the job graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$.

You can offer two ways to organize such a manager.

In a simpler embodiment, the AMOEP can be controlled by means of a specially dedicated server unit, whose function is to distribute the incoming jobs workflow $\mathbf{Z} =< Z_1, Z_2, ..., Z_M >$ between WAs. However, such a centralized organization manager AMOEP has a number of disadvantages. Firstly, a large number of WA solution to the problem of scheduling work with a single server node will be difficult because of the need to perform computationally intensive real-time receipt of assignments. Second, it is more difficult to scale AMOEPs (i.e., adding to its composition of new WA), because it is necessary to completely change not only the program of the central controller, but also the architecture of relations with the WA Manager. And finally, thirdly, the AMOEP with a central controller becomes undependable because the controller failure leads to disastrous consequences for the entire AMOEP as a whole. All these problems are much more complicated in the case where the AMOEP must not perform a single task, and the task flow into the unknown times ahead.

All these disadvantages can be avoided by using the principles of decentralized multi-agent scheduling in distributed systems [1][2][3][4]. Wherein each of the WA, included in the AMOEP must have its software agent, representing "interests" of the well during the scheduling and allocation of incoming jobs to optimize operations must be carried out between the WA by their collective interaction information through a communication channel (fig.3).

This raises the question - how agents should communicate with customers and get them jobs? Such interaction can be performed by node playing the role of "bulletin board" (BB), where customers can post their job. In this case, he job Z_l located on the (BB), must contain:

- Count $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ tasks;

- a list of vertices set $\mathbf{Q}_l$ and the operations that are attributed to them;

- moment of time $T^l_{\max}$ to which the user wishes to obtain a finished product.

Agents need to periodically poll for the purpose of loading the WA work. In this case, if the job $Z_l \in \mathbf{Z}$ it cannot be fulfilled by one WA R_i, the agents should form a certain virtual organization - community, consisting of a plurality of WA $\mathbf{R}_l =< R_i, R_j, ..., R_k >$ common purpose of which is a quest $Z_l \in \mathbf{Z}$ to set the customer in time.

In the integrated form of AMOEPs work with multiagent manager can be represented as follows.

1. The customer creates his job $Z_l \in \mathbf{Z}$ in the form of a graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$, establishes a point in time $T^l_{\max}$ to which he wants to get the finished product, and places to handle jobs.

2. Agent R_i not involved in the performance of other tasks to queries is in search of work for "their" WA. In case of the job on to $Z_l \in \mathbf{Z}$ agent attempts to join the community $\mathbf{R}_l$ for its implementation. For this agent R_i allocates tasks fragment $Z_l \in \mathbf{Z}$ which is not fixed earlier for other agents, that it can perform a set time using "own" WA. If such a fragment is detected, the agent R_i goes in $\mathbf{R}_l$ and proceeds.

4. Agent R_i monitors the progress of the operations accepted for execution fragment tasks, periodically assessing the time of their completion. If the agent, according to any reasons did not have time to complete the execution of these operations to the set point in time, it informs about it on the bulletin board and withdraws from the community $\mathbf{R}_l$.

5. In case of successful execution of all received job operation Z_l agent R_i transmits via the transport line product to another WA for the subsequent operations.

Using the principles of collective decision-making AMOEPs provides:

- high availability systems because it lacks the "bottleneck" in the form of a central controller, and the failure of any of the agents does not lead to disastrous consequences for the AMOEP as a whole;

- virtually scaling of WA consisting of AMOEP by their easy connection to the information communication channel;

- reduction of the processing load on a single software agent while solution of scheduling problem.

However, on the other hand, the use of multiagent scheduling principles in the management of AMOEP requires the development of an algorithm of the software agent. The next part of the paper is dedicated to this development.

IV. The Algorithm of Software Agent Work

Before you begin to develop multiagent scheduling algorithm AMOEP you ought to introduce the concept of "thread." Under the thread, we mean a sequence of vertices $\mathbf{H}_f =< q^f_1, q^f_2, ..., q^f_k >$ column $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ tasks $Z_l \in \mathbf{Z}$ wherein vertices q^f_j and q^f_{j+1} $(j = 1, 2, ..., k-1)$ connect arc $x(q^f_j, q^f_{j+1})$. In other words the thread defines a task set of operations Z_l to be executed sequentially. In this case under length t^p_f thread $\mathbf{H}_f$ we mean the total time, required to perform its assigned heights of operations, defined as $t_f = \sum_{i=1}^{k} \left(t_p(A^f_i) + t_{\text{п}}(S_{p,c}) \right)$, where $t_p(A^f_i)$ - time spent by WA $R_p \in \mathbf{R}$ to perform the operation A^f_i, attributed to the top $q^f_i \in \mathbf{H}_f$ $(i = 1, 2, ..., k)$;

$t_{\text{п}}(S_{p,c})$ - the time, required to transport the product from the WA $R_p \in \mathbf{R}$, performing surgery A^f_i $R_c \in \mathbf{R}$ performing on stage following operation A_{i+1}^f thread $\mathbf{H}_f$; $S_{p,c}$ - the length of the transport line between the R_p and R_c.

If surgery A_i^f and A_{i+1}^f performed the same WA R_p, respectively, $t_п(S_{p,p}) = 0$.

Obviously, if the entire thread $\mathbf{H}_f$ is performed one resource R_p^f, its length will be

$$t_f = \sum_{i=1}^{k} t_p(A_i^f) . \quad (1)$$

Thus, if the known time required T_{k+1}^f for execution of the entire thread $\mathbf{H}_f$, it is possible to determine the allowable time points T_d^f beginning of all operations A_d^f, assigned to vertices $q_d^f \in \mathbf{H}_f$ $(d = 1,2,...,k)$

(In which the WA R_p time to complete the entire thread $\mathbf{H}_f$ to the desired point in time T_{k+1}^f) as

$$T_d^f = T_{k+1}^f - \sum_{i=d}^{k} t_p(A_i^f) \quad (d = 1,2,...,k-1).$$

$$(d = 1,2,...,k-1) . \quad (2)$$

Based on these considerations, we propose the following procedure for multi-agent scheduling work AMOEPs when the task flow is fulfilled.

The user creates his task $Z_l \in \mathbf{Z}$ in the form of a graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ and determines the desired point in time $T_{\max}^l$ to which its decision is to be obtained. Descriptor represented thus job is $Z_l \in \mathbf{Z}$, placed on a bulletin board.

Agents, not involved in performing any tasks, refer to the BB in search of work. If the agent is free, RF $R_p \in \mathbf{R}$ detects on to handle the job Z_l, He makes an attempt to enter the Community $\mathbf{R}_l$ for its implementation.

Since, as we adopted above, each WA has some specialization (i.e., can perform a limited set of operations), in the general case, it may be that the WA R_p, able to perform all the operations, is not assigned to the vertices of $\mathbf{G}_l^j(\mathbf{Q}_l^j, \mathbf{X}_l^j)$ tasks Z_l . Therefore, in the graph $\mathbf{G}_l^j(\mathbf{Q}_l^j, \mathbf{X}_l^j)$ subgraph $\mathbf{G}_l^{jp}(\mathbf{Q}_l^{jp}, \mathbf{X}_l^{jp})$ must allocate whose vertices in plurality of operation $\mathbf{A}_p$ performed by RF R_p. After that it is necessary to analyze whether there are $\mathbf{G}_l^{jp}(\mathbf{Q}_l^{jp}, \mathbf{X}_l^{jp})$ peaks in the box for which is there is required time of their execution set. If there are no such peaks, it means that the agent R_p could not join the community $\mathbf{R}_l$ to fulfill tasks Z_l and so it reverts to the regime of the survey with a view to find other jobs.

Otherwise, agent R_p highlights in the column $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$, that is stored in the tag assignments Z_l on the longest thread $\mathbf{H}_1 =< q_1^1, q_2^1, ..., q_k^1 >$, according to the expression (1), the top end q_k^1 which is attributed to the desired point in time of its execution T_{k+1}^1. The latter may be performed, using one of the well-known search algorithms of extreme paths on graphs [5].

Further, the agent R_p determines in accordance with (2) at time T_1^1 when it needs to begin the first operation thread $\mathbf{H}_1$, i.e. operations A_1^1, attributed to the top $q_1^1 \in \mathbf{H}_1$ in order to have time to complete the execution of the entire thread $\mathbf{H}_1$ at a given moment of time T_{k+1}^1.

If it turns out that $T_1^1 < T_{тек}$ where $T_{тек}$ - the current time, it means that the WA R_p can not ensure that the entire thread workflow $\mathbf{H}_1 =< q_1^1, q_2^1, ..., q_k^1 >$ is proceeded to the point in time set by the customer T_{k+1}^1. Since we assumed that the execution of transactions in various WA is equal, neither any other WA also is not able to carry out this thread to the desired point in time, which indicates that the job Z_l can’t be made to the installed client in time. In this case, the agent R_p sends a message to the appropriate. The task Z_l removed from the TO, and the consumer is sent a message about the impossibility of his job to the set point in time. Thereafter well agent R_p reverts to the survey in search of work.

If the condition $T_1^1 \geq T_{тек}$ of the executed thread $\mathbf{H}_1$ is true, the agent R_p accepts the execution sequence of operations, assigned to its vertices. When this agent R_p gets a next modification of job descriptor Z_l, namely:

1. its identifier is recorded in the list of community members $\mathbf{R}_l$ to fulfill tasks Z_l;

2. vertices, belonging to the thread $\mathbf{H}_1$, are excluded from the count $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ tasks Z_l, resulting in a new graph $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1) = \mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)/\mathbf{H}_1$;

3. all the vertices of the graph $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$ incident to a vertex thread $\mathbf{H}_1$, are attributed to the required times of their execution, which are determined from the following considerations.

Suppose that some vertex q_f^2 column $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$ incident to the top q_b^1, belongs to the thread $\mathbf{H}_1$. This means that the operation is feasible, attributed to the top q_b^1 can be started only after the operation, attributed to the top q_f^2 column $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$. Therefore, it is obvious that the results of operations, attributed to the top q_f^2, should be prepared and transferred by WA R_p performing operations yarns $\mathbf{H}_1$ not later than the desired time point T_b^1 start of the WA R_p operations A_b^1, attributed to the top q_b^1, determined according to expression (2) as

$$T_b^1 = T_{k+1}^1 - \sum_{i=b}^{k} t_p(A_i^1). \qquad (3)$$

Otherwise WA R_p will not have time to finish the execution of the thread, taken $\mathbf{H}_1$ to the desired point in time T_{k+1}^1.

Therefore, the top q_f^2 column $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$, attributed to the required time of its execution $T_{f+1}^2 = T_b^1$ as well as the identifier of the WA R_p to which the operation execution results attributed to the top q_f^2 should be transferred.

Similarly the moments are determined by the required T_{m+1}^2 execution of all other vertices $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$ incident to the heights of the thread $\mathbf{H}_1$.

If, after modifying a new graph $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$ tasks Z_l on to another is not empty, i.e., $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1) \neq \varnothing$, the process of creating a community $\mathbf{R}_l$ for the assignment Z_l continues further.

Let us assume that after some time another free WA R_c detects on to handle the job Z_l and attempts to join the community $\mathbf{R}_l$ for its implementation.

For this, agent R_c highlights in the column $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$ subgraph $\mathbf{G}_l^{1c}(\mathbf{Q}_l^{1c}, \mathbf{X}_l^{1c})$ whose vertices are assigned operations included in the executable WA R_c a bunch of $\mathbf{A}_c$. Further, in the column $\mathbf{G}_l^{1c}(\mathbf{Q}_l^{1c}, \mathbf{X}_l^{1c})$ agent R_c allocates the longest thread $\mathbf{H}_2 = < q_1^2, q_2^2, ..., q_f^2 >$ the ultimate top q_f^2, which is attributed to the required runtime T_{f+1}^2, and analyzes the possibility of its execution with the help of "his" WA to this point in time. The expression (2) defines the time T_1^2 of the operation execution A_1^2, attributed to the first vertex q_1^2 of this thread $\mathbf{H}_2$ and compares it with the current time $T_{\text{тек}}$. If $T_{\text{тек}} > T_1^2$, it means that the WA R_c cannot carry out this thread to the desired point in time T_{f+1}^2 thus says that all the task as a whole can't be fulfilled in the demanded time limits. In this case, the task is removed from the TO, the customer is sent a message about the impossibility of his job to be done in desired time, and agent R_c again stands into polling mode up looking for work for "their" resource.

In case, if the condition $T_1^2 \geq T_{\text{тек}}$ is fulfilled, the agent R_c assumes the execution of transactions, attributed to the heights of the thread $\mathbf{H}_2$ and performs the next task descriptor modification Z_l on the TO:

- the identifier of the agent R_c is recorded in the list of community members $\mathbf{R}_l$ to address the problem Z_l;

- the vertices, belonging to the thread $\mathbf{H}_2$ are excluded from the count $\mathbf{G}_l^1(\mathbf{Q}_l^1, \mathbf{X}_l^1)$, resulting in a new graph $\mathbf{G}_l^2(\mathbf{Q}_l^2, \mathbf{X}_l^2)$.

- tops q_p^3 column $\mathbf{G}_l^2(\mathbf{Q}_l^2, \mathbf{X}_l^2)$ incident to the vertex q_d^2 thread $\mathbf{H}_2$, attributed ID WA R_c which results of the execution of these operations must be sent as well as the desired time of their execution, are defined as $T_{p+1}^3 = T_{f+1}^2 - \sum_{i=d}^{f} t_c(A_i^2)$.

Further, the job allocation process operates Z_l, activated by next available agent who discovered it to handle on, etc. until then, until it turns out that after the next modification of the count $\mathbf{G}_l^j(\mathbf{Q}_l^j, \mathbf{X}_l^j)$ became empty, which means that all the task operations Z_l have dismantled agents, which entered the community $\mathbf{R}_l$ for its implementation.

After some agent R_p is chosen to thread execution $\mathbf{H}_f = < q_1^f, q_2^f, ..., q_k^f >$ he starts to perform operations, assigned to vertices, using "their" WA. In this case, before starting the next operation A_d^f, attributed to the top $q_d^f \in \mathbf{H}_f$ $(d = 1, 2, ..., k)$ agent R_p must verify, first, the presence of all components and workpieces required for its implementation, and, secondly, observance of the temporary schedule of the entire thread $\mathbf{H}_f$ generally.

As for the performance of sub-tasks, A_d^f may require the blank, obtained by performing another adjacent thread WA R_c. Then at the time of start of the WA R_p operations A_d^f it may occur that these blanks are not yet available. In this case, the agent R_p should go into standby mode to necessary preparations. This expectation can last as long as the condition $T_d^f \geq T_{\text{тек}}$ is true, where T_d^f desired start time operation execution A_d, attributed to the top $q_d^f \in \mathbf{H}_f$ and determined according to the expression (2).

If it turns out that $T_d^f < T_{\text{тек}}$, it means that the WA R_p does not have time to complete the remaining operation thread $\mathbf{H}_f$ to the desired point in time T_{k+1}^f. In this case, the agent R_p must be known before that happened behind schedule the task, and therefore the task Z_l cannot be resolved to the installed customer in time. In this case, the task is removed from the TO, the user is informed about the impossibility of its execution by the time set by him, and all agents entered into the Community $\mathbf{R}_l$ of its decision send a message about the termination of the assignment process Z_l after which they go into search mode to new jobs.

After successful execution of the thread operations $\mathbf{H}_f$ agent R_p re-enters the polling mode to the purpose of entering into a new community for the next assignment.

The process of executing the job Z_l lasts as long as it appears that the list of participating community agents $\mathbf{R}_l$ to implement is empty, and the empty graph $\mathbf{G}_l^j(\mathbf{Q}_l^j,\mathbf{X}_l^j)$ is stored in the task descriptor Z_l. After that, the job is removed from the TO, and the customer is sent a message about the successful completion of his mission.

The above process corresponds to the following algorithm, describing enlarged functioning of an agent, representing WA R_p in the process of scheduling the work AMOEPs.

Algorithm 1.

1. Free WA R_p BB polls.

2. When a job is on Z_l agent R_p analyzes the task graph $\mathbf{G}_l^j(\mathbf{Q}_l^j,\mathbf{X}_l^j)$. If $\mathbf{G}_l^j(\mathbf{Q}_l^j,\mathbf{X}_l^j)=\varnothing$, the transition to the claim 1, otherwise

3. In the column $\mathbf{G}_l^j(\mathbf{Q}_l^j,\mathbf{X}_l^j)$ agent R_p stands subgraph $\mathbf{G}_l^{jp}(\mathbf{Q}_l^{jp},\mathbf{X}_l^{jp})$, whose vertices are assigned a plurality of operation $\mathbf{A}_p$ performed by WA R_p.

4.If $\mathbf{G}_l^{jp}(\mathbf{Q}_l^{jp},\mathbf{X}_l^{jp})=\varnothing$ go to claim 1, otherwise

5. Agent R_p highlights in the column $\mathbf{G}_l^{jp}(\mathbf{Q}_l^{jp},\mathbf{X}_l^{jp})$ the longest thread $\mathbf{H}_j^p=<q_1^{jp},q_2^{jp},...,q_k^{jp}>$, the top end of which is attributed to the required runtime T_{k+1}^{jp} (At the time of job placement Z_i at the required time to $T_{k+1}^{jp}=T_{\max}^l$ attributed to only the final top q_k column $\mathbf{G}_l(\mathbf{Q}_l,\mathbf{X}_l)$). If any thread in the box $\mathbf{G}_l^{jp}(\mathbf{Q}_l^{jp},\mathbf{X}_l^{jp})$ is not true, then go to 1, or

6. Agent R_p determines the allowable time when the need to start the thread $\mathbf{H}_j^p=<q_1^{jp},q_2^{jp},...,q_k^{jp}>$ to be able to complete its performance to the desired time T_{k+1}^{jp} as $T_1^{jp}=T_{k+1}^{jp}-\sum_{i=1}^{k}t_p(A_i^{jp})$.

7. If $T_1^{jp}<T_{\text{тек}}$ where $T_{\text{тек}}$ - the current time, the transition to claim 16, or

8. Agent R_p assumes execution thread $\mathbf{H}_j^p=<q_1^{jp},q_2^{jp},...,q_k^{jp}>$, which modifies the job descriptor Z_l on the TO:

recording in the list of members of the community $\mathbf{R}_l$ your ID; modifies count $\mathbf{G}_l^j(\mathbf{Q}_l^j,\mathbf{X}_l^j)$ tasks Z_l by deleting the thread vertices $\mathbf{H}_j^p=<q_1^{jp},q_2^{jp},...,q_k^{jp}>$, i.e. $\mathbf{G}_l^{j+1}(\mathbf{Q}_l^{j+1},\mathbf{X}_l^{j+1})=\mathbf{G}_l^j(\mathbf{Q}_l^j,\mathbf{X}_l^j)/\mathbf{H}_j^p$; ascribes to the heights $q_j^{j+1,p}$ modified Count $\mathbf{G}_l^{j+1}(\mathbf{Q}_l^{j+1},\mathbf{X}_l^{j+1})$ incident to the vertex $q_b^{j,p}$ thread $\mathbf{H}_j^p=<q_1^{jp},q_2^{jp},...,q_k^{jp}>$ WA ID R_p you want to transfer the results of execution of the operation $A_f^{j+1,p}$ attributed to the top $q_f^{j+1,p}$ As well as the desired time of their execution, determined according to the expression (2).

9. Agent R_p proceeds to the flowchart assigned thread vertices $\mathbf{H}_j^p=<q_1^{jp},q_2^{jp},...,q_k^{jp}>$; $d=1$.

10. If $T_d^{jp}<T_{\text{тек}}$ where $T_d^{jp}=T_{k+1}^{jp}-\sum_{i=d}^{k}t_p(A_i^{jp})$ - start time of the desired operation A_d^{jp}, attributed to the top $q_d^{jp}\in\mathbf{H}_j^p$ Then go to claim 16, or

11. Agent R_p checks for blanks, necessary to perform the operation A_d^{jp}. If the blanks are not yet available, go to n. 10, or

12. Agent R_p performs A_d^{jp} ascribed to the top $q_d^{jp}\in\mathbf{H}_j$ using its WA.

13. If the agent R_p has already received a flag of termination of the assignment Z_l, the transition to the claim 1, otherwise

14. $d=d+1$, if $d\le k$, the transition to claim 10, or

17. Agent R_p according to the successful implementation of all the operations, the thread assigned to the vertices $\mathbf{H}_j$ tasks Z_l. Identifier agent R_p is excluded from the list of members of the community $\mathbf{R}_l$ for performing this task. Transition to claim 1.

16. Assignment Z_l cannot be made to the installed client to the time $T_{\max}^l$. descriptor assignment Z_l removed from the TO, the customer is sent a message about the impossibility of his job proceeding to the desired point in time, and all agents, whose numbers are stored in the list of community members $\mathbf{R}_l$ to fulfill tasks are sent a message is sent to terminate their execution. Transition to claim 1.

V. Conclusions

The article describes the basic principles of the organization and functioning of decentralized autonomous mechatronic manufacturing plant. The implementation of these principles provides:

- a high loading of WAs in the AMOEP;

- the ability to automatically perform job flow on the manufacturing of various products;

- quasioptimal adaptive WAs allocation based on their computing power estimation in real-time when a job is in progress;

- the possibility of unlimited scalability of WAs in AMOEP due to decentralization;

- increased failureproof of AMOEP, since it lacks components, failure of which leads to failure of the entire AMOEP as a whole.

Acknowledgements

The research is supported by Russian Foundation for basic Research (RFBR) via projects 17-08-01219.

References

[1] Kalyaev A.I., I.A. Kalyaev,. 2015. *Decentralized distributed system control method of performing task flow* - Mechatronics, automation, control, № 9, s.585-598.

[2] Kalyaev A.I, I.A Kalyaev, Korovin I.S., 2015. *The method of multi-agent scheduling resources in a heterogeneous cloud environment when the task flow.* - Journal of Computer and Information Technology, number 11, pp. 34-40.

[3] Kalyaev A.I., 2013. *Multiagent Approach for Building Distributed Adaptive Computing System* / Procedia Computer Science, Volume 18, Pages 2193-2202 (URL: http://dx.doi.org/10.1016/j.procs.2013.05.390)

[4] Kalyaev A.I, Korovin I.S., 2014. *Adaptive Multiagent Organization of the Distributed Computations* / AASRI Procedia Volume 6, Pages 49-58 (URL: http://dx.doi.org/10.1016/j.aasri.2014.05.008)

[5] Reinhold E., Nivergelt Yu, N. Deo., 1980. *Combinatorial* algorithms. *Theory and practice.* - M .: Mir. - 476 p.