Автоматизовані системи управління та прилади автоматики. 2025. № 4 (187)

UDC 004.8:004.9 **DOI:** https://doi.org/10.30837/0135-1710.2025.187.142

V. Filatov, M. Chernenko

USER TASK SUPPORT IN INFORMATION SYSTEMS BASED ON AGENT TECHNOLOGIES

The **subject matter** of this paper concerns the application of multi-agent systems and software agent technologies for managing and analyzing information resources and t ask flows within distributed, heterogeneous information environments. It focuses on formal models and execution frameworks that enable intelligent control of complex agent-oriented workflows. The **goal** is to develop effective methods and models for representing, coordinating, and executing agent-oriented tasks, ensuring correctness and efficient resource use in distributed systems. The **tasks** addressed include formalizing the information space as an algebraic system comprising objects, relationships, and operations; defining elementary and functional tasks executed by users or agents; organizing interdependent tasks into flows; and modeling agent behavior via frame structures and hierarchical Petri nets incorporating metapositions and metatransitions. The **methods** employed combine algebraic system modeling, automata theory for agent behavior analysis, and extended predicate Petri nets with modifications supporting hierarchical task management. Coordination constraints and workflow correctness properties are specified using Computation Tree Logic (CTL) and dependency automata to capture inter-task dependencies and atomicity requirements. The work also applies SQL-based operations to exemplify merging of data during task execution. The **results** include a formal framework distinguishing state-based and task-based planning, a modified frame slot model enabling complex task descriptions with initial conditions and operations, and a hierarchical Petri net scheme with subordinate nets activated conditionally, thereby optimizing computations. A software agent prototype executing a sequence of conditional operations on multiple databases demonstrates the practical application of these concepts. The approach reduces execution time and computational costs by activating only necessary subordinate tasks and avoiding irrational solutions. The **conclusions** assert that the proposed agent-oriented modeling and hierarchical task execution methods successfully address the complexities of distributed information resource management and workflow control, providing a scalable means for executing fault-tolerant, atomic, and coordinated workflows. This enhances the reliability and efficiency of multi-agent systems in heterogeneous computing environments.

Keywords: multi-agent systems; hierarchical task management; Petri nets; information resource management.

Introduction

This paper considers a class of multi-agent systems for managing information resources in distributed systems. A typical representative of this class of problems is the task of planning a path to achieve a desired goal from a given initial situation. The result of solving such a problem should be an action plan – a partially ordered set of actions. Such a plan can be represented as a scenario in which the relationships between vertices are of the types "goal-subgoal", "goal-action", "action-result", and so on. Any path in this scenario that leads from a vertex corresponding to the current situation to any of the goal vertices defines an action plan. The description of situations includes both the state of the external environment and the state of the information system. Situations form certain generalized states, and actions or changes in the external environment lead to changes in the currently actualized states.

Among the generalized states, the initial states (usually one) and the final (goal) states are distinguished. All action-planning tasks can be divided into two types, corresponding to different models: state-space planning and task-space planning. The representation of problems in a state space involves specifying a set of descriptions: states, a set of operators and their effects on state transitions, and the goal states. State descriptions can take the form of strings of symbols, vectors, two-dimensional arrays, trees, lists, and so on. Operators transform one state into another. The state space can be represented as a graph, where vertices are labeled with states and arcs with operators. The state-space planning method can be regarded as a special case of planning by reduction, since each application of an operator in the state space reduces the initial problem to two simpler ones, one of which is elementary. Problem-space planning consists of the successive reduction of the original problem to increasingly simpler ones until only elementary problems remain. A partially ordered set of such problems constitutes the solution to the original problem.

Literature review and problem statement definition

Software agents emerged as one of the most significant achievements in computer science in the 1990s. The problem of intelligent agents and multi-agent systems (MAS), which has a long history, was shaped within the scope of research on distributed artificial intelligence and, in recent years, has claimed one of the leading roles in intelligent information technologies [1]. Agents are used in applied systems where humans and computers are closely interconnected in managing information processes [2]. The abundance of diverse information on software agents leads to inconsistencies in defining what constitutes a software agent. Therefore, it is first necessary to clarify the essence of the concept of a "software agent" and to identify the key properties of software agents. Below we consider the main properties and specifications of agent-oriented tasks.

Definition of an Agent-Oriented Task. An agent-oriented task in a workflow is a unit of work that can be executed by a processing entity, such as an application-level computing system or, for example, a database management system (DBMS) [3]. A task may be defined independently of the processing entity capable of executing it, or based on the capabilities and behavior of that entity. The structure of a task can be specified by defining: a set of task execution states, a set of transitions between these states, and the conditions that make these transitions permissible (transition conditions can be used to specify inter-task execution requirements). The execution order of each task is determined by specifying the set of states observable from outside and the set of transitions between these states. Furthermore, certain characteristics of processing entities can be defined that may influence the requirements for task execution [4].

Coordination Requirements for Agent-Oriented Tasks. Coordination requirements are typically expressed in terms of inter-task execution dependencies and dataflow dependencies.

Execution or Correctness Requirements. Execution requirements are defined to constrain the workflow execution in such a way that correctness criteria, dependent on the application, are met. They include requirements for failure atomicity, execution atomicity (including visibility rules that determine when the results of a committed task become visible to other concurrently running workflows), as well as requirements for synchronous execution control and recovery.

Failure Atomicity Requirements in Workflows. The traditional notion of failure atomicity is that the failure of any task causes the entire workflow to fail. However, in many cases, a workflow may tolerate the failure of a single task by, for example, executing a functionally equivalent task on another node. A system executing a workflow (a workflow scheduler) must ensure that any execution completes in a state that satisfies the failure atomicity requirements specified by the developer. We refer to such states as acceptable workflow termination states. All other execution states belong to the set of unacceptable termination states, in which failure atomicity may be violated.

An acceptable termination state can be either committed or aborted. A committed acceptable termination state is an execution state in which the goals of the workflow have been achieved. Conversely, an aborted acceptable termination state is a correct termination state in which the workflow has failed to achieve its goals. If an aborted acceptable termination state is reached, all undesirable effects of partial workflow execution must be eliminated in accordance with the failure atomicity requirements [5–7].

Tasks are modeled by their states together with significant events corresponding to transitions between states – start, commit, rollback, etc., – which may be enforced, rejected, or delayed. Inter-task dependencies, such as ordering dependencies and existence dependencies between significant task events, are formally defined using Computation Tree Logic (CTL) and have corresponding dependency automata that can be automatically constructed. To address the aforementioned characteristics in user tasks, it is advisable, at the first stage, to consider issues related to the information space and, specifically, to the formal representation of such a space.

Research objective. The objective of this work is to develop effective models and methods for managing and analyzing task flows within a unified heterogeneous information space, based on software agent technology.

Research materials and methods

Development of a model of the distributed system's information space

The inf ormation space of a distributed information system can be represented as an abstract algebraic system

$$P =< O,\ S,\ \Omega >, \qquad (1)$$

where O – the objects of the information space; S – the relationships between objects O; Ω – the set of operations for manipulating objects within the space P.

The objects in model (1) can include components of a modern computing system – files of all types, directories, logical and physical disks, personal computers, and so forth. The relationship $s_i \in S$ between the objects of the information space defines a specific configuration of the computing environment $p_i =< o_i, s_i, q_i >$, where $p_i \in P$, $o_i \in O$, $q_i \in \Omega$, oriented towards a specific user or group of users $u_i \in U$, $i = 1, N$, where N is the number of users.

Depending on the goals set by the user and the computing environment, all actions are performed based on operations from Ω.

All operations on objects, according to the defined goals, are initiated by a user $u_i \in U$ of the information space, following either a formalized action plan $x_{i1}^*(t), x_{i2}^*(t), \ldots x_{im}^*(t)$ or a unitary action $x_i^*(t)$.

In the general case, the i-th action plan $x_i(t)$, if executed as a one-time interaction between the user and the information environment (such as viewing the contents of a file, moving a file within the computing environment, etc.), can be interpreted as an elementary task that a user $u_i \in U$ solves with the help of the information system $P(O, S, \Omega)$.

The result of solving an elementary task $x_i(t)$ can be defined as a mapping $z_i : x_i \Rightarrow y_i$, the execution of which requires an operation q_i, $i \in I$, where $y_i \in Y$ is the solution to the task $x_i(t)$. An elementary task, within the framework of the proposed approach, can be represented as $z_i = (o_i, s_i, q_i, x_i(t), y_i)$.

Elementary tasks can be combined into functional tasks by merging elementary operations into a sequence of interrelated operations forming an algorithm $A_k = \{q_1, q_i, \ldots q_m\}$, $i = \overline{1, m}$, $A_k \in A$, $k \in K$ where A is the set of solutions to functional tasks and K is the number of functional tasks.

Depending on the structure and interdependencies of the tasks solved by the user, they can be organized into task flows. A task flow $A_k \in A$ is defined as a sequence of tasks for which the following condition is true: $y_{ij}(t) = p_{ij}(x_{ij}(t))$, $i \in I$, $j \in K$, considering the constraint $x_{ij}(t) = y_{ij}(t)$, $i \neq j$, meaning that the result of solving the preceding task serves as the input for executing the subsequent task. The interrelationship scheme of subtasks in a flow is shown in Fig. 1.

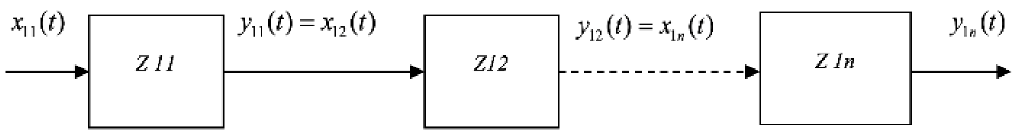

Fig. 1. The interrelationship scheme of subtasks in a flow

To implement the interaction technology of a software agent with objects of a heterogeneous computational environment, a frame structure can be used as a model of the software agent.

A slot in such a model is a logical construct designed to perform specific tasks assigned to the frame – that is, to the software agent. Slots in the frame may be removed, added, or reassigned to different functional roles.

Formal Model of a Software Agent

Representing the logical model of a software agent in the form of a frame makes it possible to effectively apply automata theory for constructing and analyzing the model of agent behavior.

In general, such a model can be expressed as follows:

$$FR\left\{\left\langle R_1,\ C_{11},\ C_{12},\ \ldots\ C_{1m}\right\rangle;\ \left\langle R_2,\ C_{21},\ C_{22},\ \ldots\ C_{2m}\right\rangle,\ \left\langle R_{k1,}C_{km}\right\rangle\right\}, \qquad (2)$$

where FR – the name of the frame; $\left\langle R_i, C_i\right\rangle$ – the i-th slot of the frame; R_i – the slot name; C_i – the slot value.

A slot in model (2) is a logical construct for performing specific tasks of the frame (software agent). Slots can be deleted, added, or reassigned. However, in the form (2), the classical slot representation as $\langle name\rangle$, $\langle value\rangle$ cannot fully reflect the requirements of a logical model of a software agent [8, 9].

We modify the structure of the slot and present it in the following form:

$$SLOT = Y\left\langle,\ D,\ dom,\ r_i,\ Q,\ W\right\rangle, \qquad (3)$$

where Y – a set of attribute names; D – a set of domains; dom – a $Y \to D$ mapping; r_i – a tuple-model of the agent's i-th task; W – a set of operations over relations; $r_i = \{R_i\}$, where $\{R_i\}$ – a set of states of the tuple-model r_i; Q – a set defining the initial conditions and criteria for task execution.

A slot may be designed using a typical set of attributes $Y = \left\{\langle OBJ\rangle,\ \langle ACT\rangle,\ \langle CON\rangle,\ \langle STA\rangle\right\}$. Table 1 lists possible basic attribute types.

Table 1. *Basic attribute types*

Entity	**Name**	**Description**
OBJECT	OBG	Database, file, folder, disk, PC
ACTION	ACT	Copy, monitor, protect
CONDITION	CON	IF–THEN predicate

STATUS	STA	1 – action executed successfully;
		0 – action not executed;
		* – indeterminate result

In the proposed "frame-software agent" model, n slots form a finite number of internal agent states. Each slot solves one task in the information environment and is associated with exactly one action. Each action elicits a reaction signal from the environment.

The behavior of the software agent, consistent with its functional goals, can be implemented, for example, using the theory of finite-state machines.

Example. Task for a software agent. Every day at 18:00, copy the file Itog.txt from directory C:\PR to directory C:\ARXIV. The structure of the software agent is shown in Table 2.

Table 2. *Agent's structure*

OBG	CON	ACT	STA
C:\PR\Itog.txt	Time = 18:00	Copy from C:\PR to C:\ARXIV	1

Network Model for Representing and Analyzing Interrelated Tasks

As a visual means of representing a task flow, a network model can be proposed. It provides a graphical representation of elementary tasks and their interrelations, whose execution is required to complete the entire task flow. In this model, the network consists of directed arcs connecting pairs of nodes. Arcs (edges) represent elementary tasks characterized by computational resource costs. Nodes (depicted as circles) represent events, i.e., strictly defined time instants.

For example, an event may represent the moment when all database operations (insert, update, edit) are completed, enabling a file-copy operation into an archive directory.

Formally, such a task flow can be represented as:

$$A_1 = \{q_1,\ q_2,\ q_3,\ q_4\},$$

where A_1 – a task flow name; q_1 – a database merge operation; q_2 – an update operation on a file attribute; q_3 – a logical operation identifying user exit from edit mode; q_4 – a database file copy operation to the archive. Fig. 2 illustrates a fragment of such a network model.

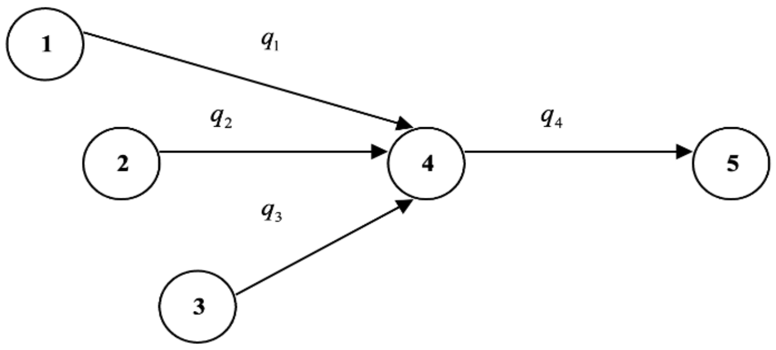

Fig. 2. A network model representing a task flow

Representing the workflow as a node–task model has significant advantages compared to the node–event method. Here, nodes represent tasks, and arcs only indicate precedence relations. There is no need to introduce the concept of an event, which simplifies network construction. Each task is uniquely associated with a node, and task execution is characterized by its duration. This approach provides access to important characteristics:

- earliest start and finish times of tasks;
- latest start and finish times of tasks;
- total time reserve.

A particular interest lies in managing and analyzing information resources with a stochastic object structure. This class of problems is analyzed using GERT systems (Graphical Evaluation and Review Technique).

A stochastic network is defined as one where the execution time of each operation follows a probability distribution [10]. Execution of a task at a node is not necessarily required for all incoming arcs. Nodes represent system states. Arcs represent transitions between states, corresponding to generalized operations defined by execution probabilities and distribution densities. Each node in a stochastic network thus performs both input and output functions.

Example. In a regional emergency management system, decision-support tasks are implemented through autonomous scenario analysis of possible emergency cases (Fig. 3).

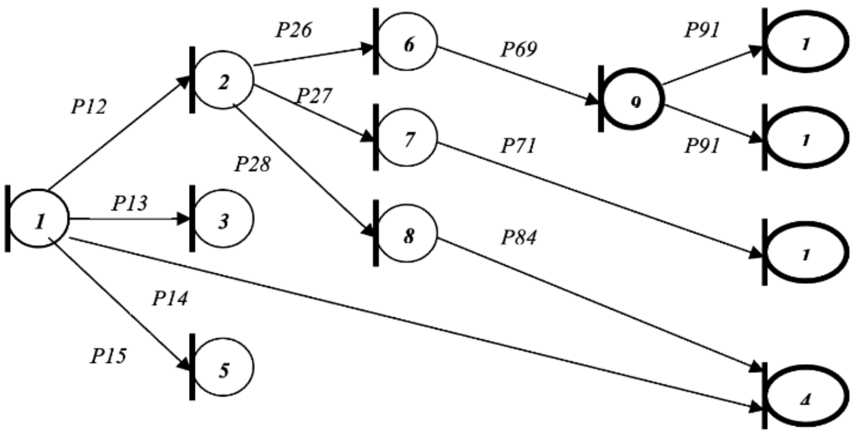

Fig. 3. A stochastic network example

Node 1: The duty officer receives probabilistic emergency alerts (fires, floods, industrial accidents, or false alarms → nodes 2, 3, 5, 4 correspondingly). Probabilities satisfy $p12 + p13 + p14 + p15 = 1$.

Assume, the situation is classified as a "Fire" by preliminary data, it is leading to scenarios: high danger (node 6), medium danger (node 7), or manageable by standard crew (node 8). Here $p26 + p27 + p28 = 1$.

The next nodes layer for scenario is build on top of deterministic nodes (bold in the diagram). They represent information support tasks, e.g., loading of databases (node 9), SQL queries for backup forces and evacuation sites (nodes 10, 11), and assessment of available local resources (node 12). By defining network parameters, one can estimate the readiness of the emergency system to respond to evolving crises [11].

High-Level Network Representation of Agent-Oriented Tasks

The design and operation of automated decision-support systems under strict resource constraints face difficulties in guaranteeing required functional and operational properties. Existing diagnostic tools mainly provide state monitoring and partial task execution control. Performance depends largely on application quality, complicated further by distributed data processing and human factors during system design.

An alternative formalism for systemic situational analysis is Petri nets [13, 14]. Petri nets allow modeling of parallel and distributed algorithms, including conflict and undesired scenarios. Formally, an algorithm graph scheme is a directed graph: $G = \langle V, E \rangle$, where V – the set of vertices and E – the set of arcs.

Vertices are of three types:

Operator vertices describe actions (including mandatory start and end).

Choice vertices describe conditional branching ("yes"/"no"), has one incoming and two labeled outcoming arcs.

Auxiliary vertices contain 3 groups:

– parallel branching (one input → many outputs) describe parallel execution of independent actions;

– parallel merging (many inputs → one output, requiring all);

– conditional merging (many inputs → one output, requiring any one).

An important advantage of graph-schemes is that the algorithmic graph-scheme can be formally transformed into a Petri net of a special type – namely, a free-choice Petri net. In this case, the justification of the correctness of the algorithmic graph-scheme reduces to verifying the correctness and safety of the corresponding Petri net. In free-choice Petri nets, each position (condition) that enables more than one transition is the sole enabling position for those transitions. The structural properties of free-choice Petri nets made it possible to develop efficient algorithms for their semantic analysis directly based on the network structure.

However, modeling control algorithms by means of algorithmic graph-schemes (free-choice Petri nets) has a number of significant drawbacks. In a graph-scheme, the algorithmic structure is predetermined and does not allow, in particular, dynamic parallelization of a task. Furthermore, the graph-scheme of an algorithm does not possess a modular structure, which makes the modeling and analysis of large-scale tasks rather difficult. To overcome this limitation, one may employ an extension of the standard Petri net formalism – nested Petri nets.

Nested Petri nets in modeling and analysis of complex interrelated processes

In nested Petri nets, tokens marking positions are regarded as objects possessing autonomous behavior, which, in turn, is also described by certain Petri nets. The term "nested nets" indicates that elements of the nets themselves are nets, similar to how, in a system of nested sets, the elements of a given set may themselves be sets. Nested Petri nets are convenient and powerful tool for modeling and analyzing hierarchical multi-agent distributed systems. They possess an inherent mechanism of modularity. Modular Petri nets can be considered as a special case of nested Petri nets.

Compared with other extensions of the Petri net formalism, owing to the concept of object and it’s construction, nested Petri nets preserve such important properties of standard Petri nets as simplicity, clarity of representation, and decidability of certain properties crucial for verification [14].

A nested Petri net consists of a system (root) net and a set of element nets representing the tokens of the system net.

In the simplest case of a two-level nested net, the element nets are ordinary Petri nets, in which tokens, as usual, are depicted as black dots, have no internal structure, and are indistinguishable from one another.

The behavior of a nested Petri net involves four types of steps:

Transfer step: firing of a transition in the system net according to the standard rules for high-level Petri nets, where element nets are treated as tokens without internal structure. A transfer step may move, create, or remove objects, but cannot alter their internal state.

Element-autonomous step: alters only the internal state (marking) of an element net without changing its location in the system net. This step is also executed according to the standard transition-firing rules of a Petri net.

Horizontal synchronization step: simultaneous firing of two transitions in two element nets located within the same position of the system net. Transitions that must fire synchronously are marked with mutually complementary labels from a special set of labels for horizontal synchronization.

Finally, *Vertical synchronization step*: used to synchronize a transition in the system net with certain transitions of the element nets. Transitions that must fire synchronously are labeled with marks from a special set of vertical synchronization labels. In this case, the label of a transition in the system net and the label of the corresponding transition in the element net must be mutually complementary.

The element nets that are moved from the preconditions of the transition in the system net during its firing are called the engaged element nets. Vertical synchronization thus means simultaneous firing of the transition in the system net and the transitions (marked with complementary labels) in the engaged element nets.

When solving problems of controlling distributed multi-agent systems, the complexity of control algorithms increases substantially, which in turn raises the importance of modeling various scenarios of system behavior under different management strategies. This creates the need for tools capable of building illustrative models of such systems’ behavior and supporting automatic verification of their semantic (behavioral) properties.

Nested Petri nets possess the following properties that make them a convenient tool for modeling and analyzing control algorithms in multi-agent systems:

− they feature a hierarchical and modular structure, which allows the model to clearly reflect the hierarchical and modular structure of the algorithm;

− element nets in a nested Petri net have their own structure and behavior, making them well-suited for modeling agents in a multi-agent system;

− horizontal synchronization of element nets corresponds to agent-to-agent interaction, while vertical synchronization models agent actions that modify the state of the environment external to those agents.

From the foregoing, it follows that nested Petri nets offer sufficiently rich expressive capabilities.

At the same time, being an extension of standard Petri nets, they preserve their advantages of simplicity and clarity of representation. Moreover, for nested Petri nets, certain properties essential for verification remain decidable [15].

Hierarchy of Networks of Interconnected Subtasks with Variable Structure

Let us consider the specific aspects of constructing networks and managing the dynamics of modeling data processing processes through the use of a combination of modified predicate nets and modified E-nets. Taking into account the known difficulties in interpreting and modeling data processing processes using predicate nets, it is deemed expedient to apply the following modification of nets:

$$S_{PR} = \left\langle P,\ T,\ F,\ A_t,\ C,\ \{V_s\},\ K,\ M_0 \right\rangle, \tag{4}$$

where P – a set of positions; T – a set of transitions; F – an incidence function between positions and transitions; C – a color function of tokens; A_t – a time parameter assigned to all network components $P,\ T,\ F, M_0$; V_s – an enabling condition of transitions $t_k \in T$, $k \in K$ related to the network components: $V_{p_k{}^I}$ – an enabling condition with respect to its input positions; $V_{p_k{}^O}$ – a condition of enabling $t_k \in T$, $k \in K$ with respect to its output positions; V_{t_k} – an enabling condition $t_k \in T$, $k \in K$ directly related to the transition; V_{Mp_l} – an enabling condition $t_k \in T$, $k \in K$ related to the marking of the incident positions, $l \in L$; V_F – an enabling condition $t_k \in T$, $k \in K$ associated with the arcs between input/output positions; K – the token capacity of positions considering C; M_0 – the initial marking vector.

The enabling condition of transition V_S also includes such properties as reliability, complexity, cost, and the degree of credibility of a specific event.

The analysis of possible solutions has demonstrated that, for the net (4), in the tasks of modeling subprocesses based on S' networks, it is advisable to apply interpreted metapositions. This is due to the fact that the positions themselves interpret the enabling conditions of the actions of simulated events and, at the same time, define the state space of the model [16–18].

Example. A resource of type A_1 is provided by information support and stored in database files, e.g., $B_{11}, B_{12},\ \ldots\ B_{1n},\ n \in N,$ where N is the number of databases. The search condition for the considered fragment of a decision-support system is formulated as follows: to determine the specified quantity of resource type $A_1 = A_1^*$ over the set $\{B_{11}, B_{12},\ \ldots\ B_{1n}\},\ n \in N$. The search of a solution is performed through two types of actions: at the first step, a SEARCH over database B_1 is executed; if the condition is not satisfied, a UNION of databases B_1 and B_2 is carried out and the search is repeated.

The procedure of searching and merging databases can be carried out in terms of the Structured Query Language (SQL), using the SELECT, JOIN, and INSERT options:

```
SELECT *
FROM b1
JOIN b2 ON b1.id = b2.id;
```

Let us represent a fragment of the above problem in the form of a *modified predicate Petri net S* (Fig. 4).

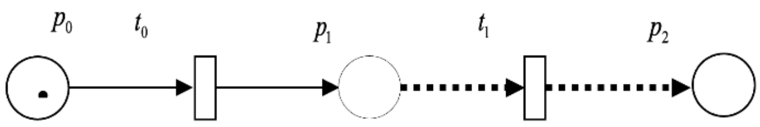

Fig. 4. *S* Petri net

The set of positions $\{p_0, p_1\}$ interpret, respectively, the input and output conditions of executing the action represented by transition t_0. The set of positions $\{p_1, p_2\}$ interpret, respectively, the input and output conditions of executing action t_1.

The initial marking vector $M_0 = (1,\ 0,\ 0)$ defines the initial state space of the net.

Description of the main constraints shown in Fig. 4. $V_{p_k{}^I}$ is the input condition of transition $t_1 \in T$, $k \in K$; $V_{p_k{}^O}$ is the output condition of transition t_1. If the input condition $V_{p_k{}^I} \neq V_{t_k}$ of transition t_1 is not satisfied, then for the given transition $t_1 \in T$, $k \in K$, a subordinate net (S') is generated (Fig. 5), where position p_1 serves as a metaposition.

The initial marking vector $M_0 = (1,\ 0,\ 0,\ 0)$ defines the initial state space of this subordinate net. The marking of position p_{13} and the execution of transition t_{13} lead to the marking of the metaposition p_1, which activates the execution of S net.

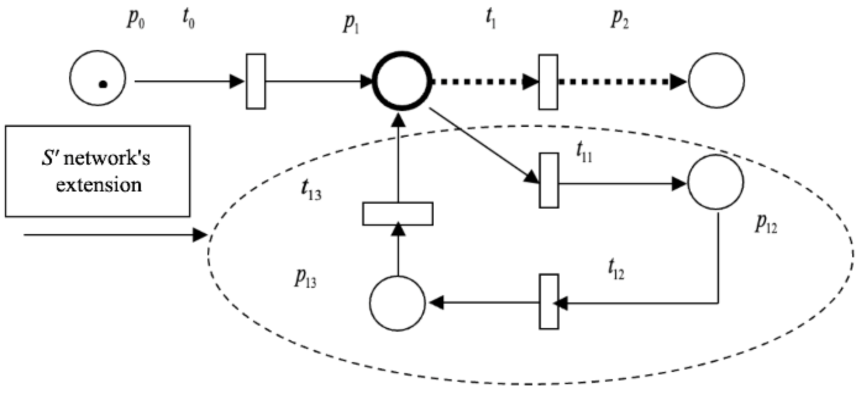

Fig. 5. S Petri net with an S' extension

If the desired solution cannot be obtained by the means of $S \cup S'$, a hierarchy of tasks is constructed based on searching for solutions from the set $\{B_{11}, B_{12}, \dots B_{1n}\}$, $n \in N$ through merging databases and extending the S' net.

Let us present the above problem as a conceptual task for a software agent:

Table 3. *Structure of the software agent "AGENT"*

ID	OBG:	CON:	ACT:	STA:
S1	C\BASE\b1.mdb	Time=10-00	Select sum([A1]) from b1.dbf	IF A1=A1* is TRUE END ELSE S2
S2	C\BASE\b1, b2.mdb	***	Join b1, b2 into b1; Select sum([A1]) from b1.dbf	IF A1=A1* is TRUE END ELSE S3
S3	C\BASE\b1, b2, b3.mdb	***	Join b1, b2, b3 into b1; Select sum([A1]) from b1.dbf	IF A1=A1* is TRUE END ELSE END

The software agent "**AGENT**" implements the task execution as a sequence of three consecutive actions in the form of operations $S1$, $S2$, and $S3$ with databases $b1$, $b2$, and $b3$. If the task condition $A_1 = A_1^*$ is satisfied at stage $S1$, the overall task is considered completed. Otherwise, an operation is executed in the form of merging two databases ($b1$ and $b2$), and the query with condition $A_1 = A_1^*$ is repeated, and so on, for up to three iterations. The software agent technology proposed in this paper allows one to successfully and efficiently solve a complex of tasks of managing and analyzing information within a unified heterogeneous information space.

Conclusions

As a result of the analysis of methods for managing information resources in corporate systems, an approach to the representation of agent-oriented tasks has been considered, based on analyzing the state of the external environment and the state of the information system. Two main methods of representing agent-oriented tasks have been investigated: state-based planning and task-based planning. The specifics of managing the state space and the structure of modified predicate nets have been analyzed in the context of hierarchical interaction of tasks.

The results obtained can be extended to a certain hierarchy of subordinate nets, taking into account their features and the necessity of generating metapositions and metatransitions. The proposed approach makes it possible to reduce time and computational resources through the activation of subordinate tasks, identification of stable dependencies, and exclusion of irrational and unsatisfactory solutions from consideration.

References

1. Wooldridge, M. (2002), "An Introduction to Multi-Agent Systems". *John Wiley & Sons Ltd*, 366 p.
2. Ponomarenko, L. A., Tsybulnyk, Ye. Ye., Filatov, V. A. (2023), "Agent technologies in information search and decision-making tasks". *USiM: Control systems and machines*, No. 1, P. 36–41.
3. Nagata, T., Ohono, M., Kubokawa, J., Sasaki, H., Fujita, H. (2002), "A multi-agent approach to unit commitment problems". *Proc. IEEE PES Winter Meet.*, P. 64–69.
4. Filatov, V. O., Yerokhin, M. A. (2023), "Improved multiobjective optimization in business process management using R-NSGA-II". *Radio Electronics, Computer Science, Control*, No. 3(187). DOI: https://doi.org/10.15588/1607-3274-2023-3-18

5. Konios, A., Khan, Y. I., Garcia-Constantino, M., Lopez-Nava, I. H. (2023), "A Modular Framework for Modelling and Verification of Activities in Ambient Intelligent Systems". *Lecture Notes in Computer Science*, Vol. 14029. Springer, Cham. DOI: https://doi.org/10.1007/978-3-031-35748-0_35
6. Pokorný, J., Richta, K., Richta, T. (2018), "Information Systems Development via Model Transformations". *Lecture Notes in Computer Science*, Vol. 10751. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-75417-8_63
7. Ziegler, P., Dittrich, K. R. (2007), "Data Integration". *Problems, Approaches, and Perspectives, Conceptual Modelling in Information Systems Engineering*, P. 39–58. DOI: 10.1007/978-3-540-72677-7_3
8. Nagata, T., Nakayama, H., Utatani, M., Sasaki, H. (2002), "A multi-agent approach to power system normal state operation", *Proc. IEEE/Power Eng. Soc. General Meeting*, Vol. 3, P. 1582–1586.
9. Solanki, J. M., Khushalani, S., Schulz N. N. (2007), "A multi-agent solution to distribution systems restoration". *IEEE Trans. Power Syst*., Vol. 22, No. 3, P. 1026–1034.
10. Martin, N., Depaire, B., Caris, A. (2016), "The use of process mining in business process simulation model construction: structuring the field". *Business & Information Systems Engineering*, Vol. 58, No. 1, P. 73–87.
11. Pourbafrani, M., Jiao, S., van der Aalst W. M. P. (2021), "SIMPT: Process improvement using interactive simulation of time-aware process trees". *Proceedings of RCIS 2021, Lecture Notes in Business Information Processing (LNBIP)*, P. 588–594.
12. Girault, C., & Valk, R. (2003), "Petri Nets for Systems Engineering". *Springer Berlin Heidelberg*. DOI: https://doi.org/10.1007/978-3-662-05324-9
13. Murata, T. (1989), "Petri nets: Properties, analysis and applications". *Proceedings of the IEEE*, No. 77(4), P. 541–580. DOI: https://doi.org/10.1109/5.24143
14. Medina-Garcia, S., Medina, J., Montaño, O., Gonzalez-Hernandez, M., Hernánndez Gress, E. (2023), "A Petri net approach for business process modeling and simulation". *Applied Sciences*, Vol. 13, 11192. DOI: 10.3390/app132011192
15. Alves, F., Merlim, R., de Carvalho, L., Souza, F. (2023), "BPMN and Petri Nets: A case study of process optimization in a project engineering company". *Proceedings of the 7th International Academic Research Conference on Engineering, IT and Applied Sciences*. DOI: 10.33422/7th.iarmea.2023.07.124
16. Qin, J., Zhao, N., Xie, Z., et al. (2017), "Business Process Modelling based on Petri nets". *MATEC Web of Conference*, Vol. 139, No. 1, P. 105–113.
17. Li, Z., Ye, Z. (2021), "A Petri Nets Evolution Method that Supports BPMN Model Changes". *Scientific Programming*, Vol. 2021, Issue 3, P. 1–16.
18. Van Hee, K. M., Sidorova, N., van der Werf, J. M. (2013), "Business process modeling using Petri nets". *Transactions on Petri Nets and Other Models of Concurrency VII*, Vol. 7480, P. 116–161. DOI: https://doi.org/10.1007/978-3-642-38143-0_4
19. Kucherenko, I., Filatov, V. O. (2004), "Decision-making models in complex systems analysis problems based on high-level networks". *Information processing systems*, No. 5, P. 139–148.

Received (Надійшла) 19.08.2025
Accepted for publication (Прийнята до друку) 07.11.2025
Publication date (Дата публікації) 28.12.2025

About the Authors / Відомості про авторів

Filatov Valentin – Doctor of Sciences (Engineering), Professor, Kharkiv National University of Radio Electronics, Professor of Artificial Intelligence Department, Kharkiv, Ukraine; e-mail: valentin.filatov@nure.ua; ORCID ID: https://orcid.org/0000-0002-3718-2077

Chernenko Mykola – PhD (Engineering Sciences), Kharkiv National University of Radio Electronics, Assistant of Artificial Intelligence Department, Kharkiv, Ukraine; e-mail: mykola.chernenko@nure.ua; ORCID ID: https://orcid.org/0009-0006-0623-5056

Філатов Валентин Олександрович – доктор технічних наук, професор, Харківський національний університет радіоелектроніки, професор кафедри штучного інтелекту, Харків, Україна.

Черненко Микола Володимирович – кандидат технічних наук, асистент кафедри штучного інтелекту, Харківський національний університет радіоелектроніки, Харків, Україна.

ПІДТРИМКА ЗАВДАНЬ КОРИСТУВАЧА В ІНФОРМАЦІЙНИХ СИСТЕМАХ НА ОСНОВІ АГЕНТНИХ ТЕХНОЛОГІЙ

Предметом роботи є застосування мультиагентних систем і технологій програмних агентів для управління й аналізу інформаційних ресурсів і потоків завдань у розподілених гетерогенних інформаційних середовищах. Розглянуто формальні моделі та межі виконання, що забезпечують інтелектуальний контроль складних агентно-орієнтованих робочих процесів. **Мета дослідження** – розробити ефективні методи й моделі подання, координації та виконання агентно-орієнтованих завдань, що гарантують коректність і ефективне використання ресурсів у розподілених системах. **Завдання**: формалізація інформаційного простору як алгебраїчної системи; визначення елементарних і функціональних завдань; організація взаємозв’язків між ними в потоки завдань; моделювання поведінки агентів за допомогою структур фреймів та ієрархічних мереж Петрі з метапозиціями й метапереходами. **Методи** основані на формальному моделюванні інформаційного простору, впровадженні теорії автоматів для аналізу поведінки агентів, використанні розширених предикатних мереж Петрі з метою підтримки ієрархічного управління завданнями, а також на застосуванні логіки обчислювального дерева (CTL) й автомата залежностей для формального визначення взаємозалежностей і вимог до узгодженості виконання. Як приклад, використано операції SQL для ілюстрації злиття баз даних у процесі виконання завдань. **Результати дослідження** передбачають: формалізовану структуру завдань агентів із чітким розмежуванням планування в просторі станів і завдань, удосконалену модель слотів у фреймі для комплексного опису атрибутів завдань, ієрархічний каркас мереж Петрі з підлеглими мережами для оптимізації обчислень і реалізацію програмного агента, який послідовно виконує умови із злиттям баз даних для досягнення цілей. Це дає змогу скоротити час виконання й обчислювальні ресурси внаслідок активації тільки необхідних підзавдань і вилучення неефективних рішень. **Висновки** підтверджують, що запропоновані модель і методи ефективно розв’язують складні завдання управління розподіленими інформаційними ресурсами й контролю робочих процесів, забезпечуючи масштабованість, надійність й атомарність виконання в мультиагентних системах гетерогенних обчислювальних середовищ.

Ключові слова: мультиагентні системи; ієрархічне управління завданнями; мережі Петрі; управління інформаційними ресурсами.

Bibliographic descriptions / Бібліографічні описи

Filatov, V., Chernenko, M. (2025), "User task support in information systems based on agent technologies", *Management Information Systems and Devises,* No. 4 (187), P. 142–155. DOI: https://doi.org/10.30837/0135-1710.2025.187.142

Філатов В. О., Черненко М. В. Підтримка завдань користувача в інформаційних системах на основі агентних технологій. *Автоматизовані системи управління та прилади автоматики*. 2025. № 4 (187). С. 142–155. DOI: https://doi.org/10.30837/0135-1710.2025.187.142