11h International Conference “Distributed Computing and Grid Technologies in Science and Education” (GRID’25),
10 July 2025

ARCHITECTURE, ANALYSIS AND DESIGN OF AN ADAPTIVE MULTI-AGENT INTELLIGENT SYSTEM

Elena Vyacheslavovna Nurmatova
Evgeny Igorevich Zaitsev

Key aspects AMAIS

1. The basis is a set of interacting intelligent agents

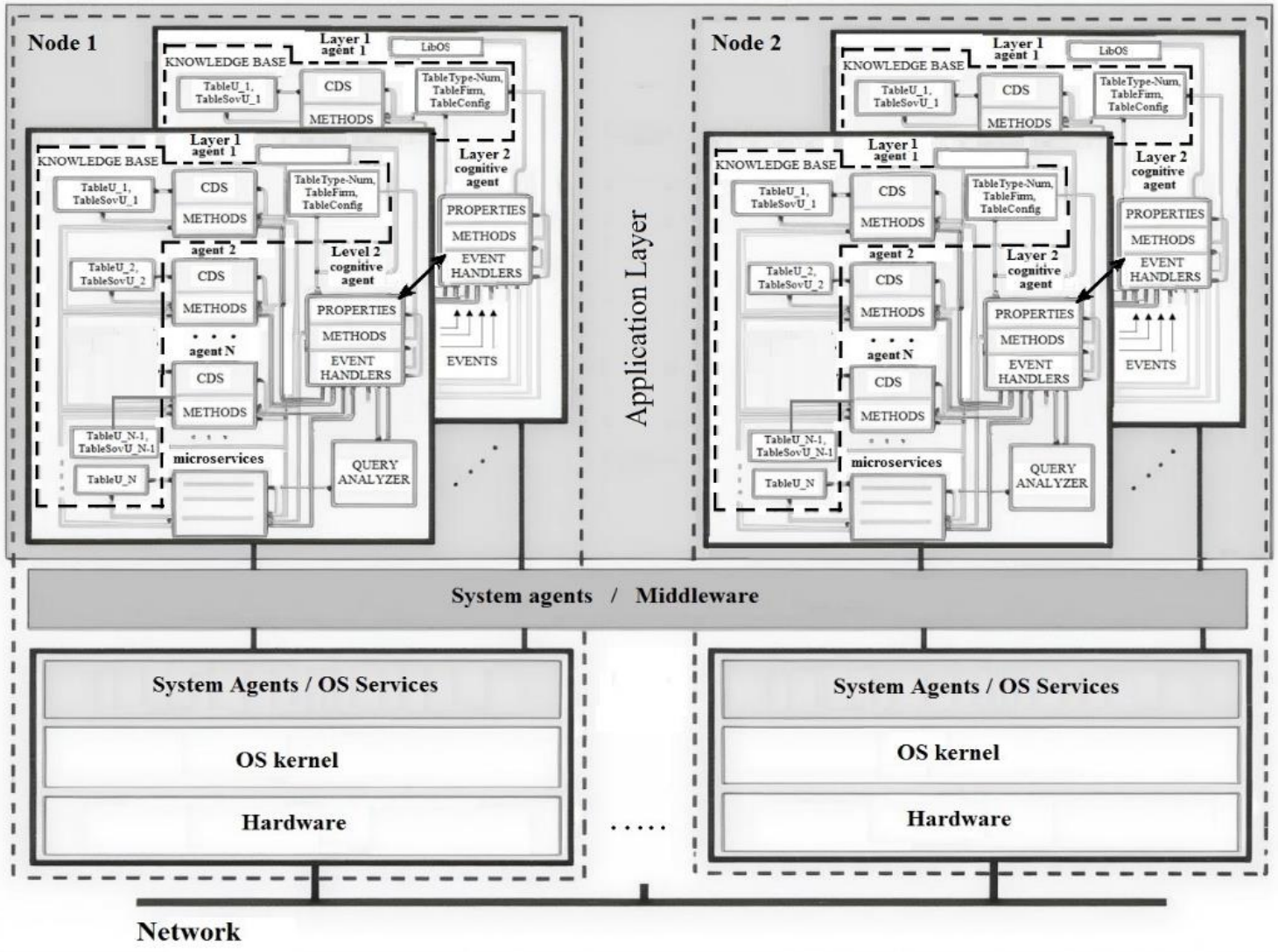
2. The essence of adaptability is the ability to dynamically change.

3. The behavior of agents (strategies, rules, decision-making policies, parameters), the structure of interactions (communication patterns, coalitions, agent roles), the composition of the system (adding/removing agents) are changing.

4. The causes of changes in the external environment are possible due to new threats, constant and dynamic data updates, and changes in time and hardware resources. There may also be changes within the system (agent failures, conflicts, load changes), or changes in goals and success criteria.

5. The purpose of adaptation is to maintain or improve efficiency, reliability, sustainability, and the ability to achieve goals in an environment of uncertainty and variability.

Architecture of an adaptive multi-agent intelligent system



Intelligent software agents solve applied tasks using a knowledge base. They interact with each other, as well as with microservices and system agents that monitor system resources and balance the load in case of a large number of requests.

Modeling multi-agent system using the python-framework pyMAS

- to implement an agent with first-order dynamics

```python
from pymas.agent import Agent

# Create my custom Agent class called MyAgent
class MyAgent(Agent):

    def __init__(self, ni=1, no=1, ns=2, f=None, *, tStart=0, \
                 init_states=None, index: int=None):
        Agent.__init__(self, ni, no, ns, f, tStart=tStart, \
                       init_states=init_states, index=index)

    # My custom dynamics: Single integrator: x_dot(t) = u(t) => f = u
    def f(self, t, x, u):
        return u

    def output(self):
        return
```

- to connect the agents, we define the network adjacency matrix

```python
from pymas.network import Network

# Create an instance of Network with intended topology:
#    - Define Adjacancy matrix:
A = np.array([[0, 1, 1, 0, 0, 0],
              [1, 0, 1, 1, 0, 1],
              [1, 1, 0, 0, 1, 0],
              [0, 1, 0, 0, 0, 1],
              [0, 0, 1, 0, 0, 1],
              [0, 1, 0, 1, 1, 0]])
#    - Create the network instance:
net = Network(A, listOfAgents)
```

pyMAS

Modeling multi-agent system using the python-framework pyMAS

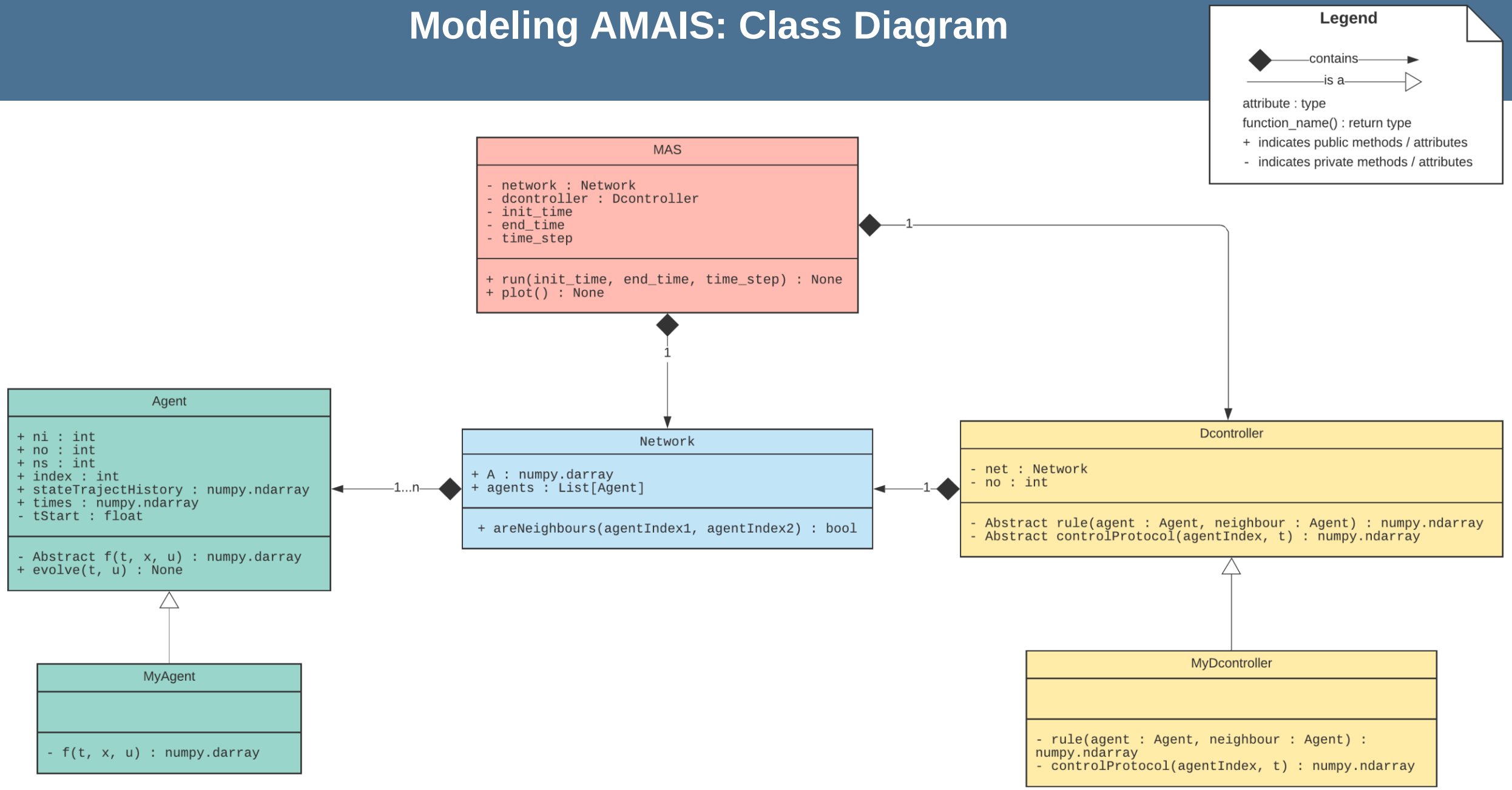
- **to implement the distributed control law**

```python
from pymas.dcontroller import Dcontroller

# Create a custom distributed control strategy by implementing Dcontroller methods:
class MyDcontroller(Dcontroller):

    def __init__(self, net: Network):
        Dcontroller.__init__(self, net)

    def rule(self, agent, neighbour):
        return neighbour.stateTrajectHistory[-1] - agent.stateTrajectHistory[-1]

    def controlProtocol(self, agentIndex: int,  t): # Simple sigma protocol
        # # DEBUG:
        # print("For agent ", agentIndex, "-- t: ", t)

        # # if t == 0 then return zero output vector (since u(0) is 0):
        # if t == self.net.agents[agentIndex].tStart:
        #     return np.zeros(shape=(self.ni, 1))
        # Calculate the control input of "agentIndex"-th agent:
        u = np.zeros(shape=(self.ni, ))
        for a in self.net.agents:
            if self.net.areNeighbours(a.index, agentIndex):
                u += self.rule(self.net.agents[agentIndex], a)
        return u

# Create the Dcontroller instance:
dcont = MyDcontroller(net=net)
```

- **to run a simulation with a specified agent**

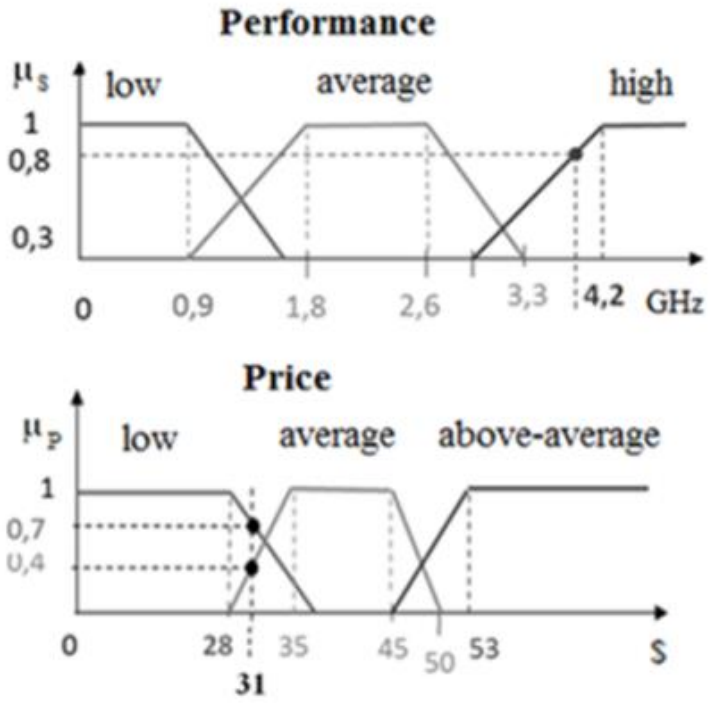
```python
from pymas import mas

# Create a MAS instance
mas = mas.MAS(network=net, dcontroller=dcont)

# Run the simulation from t=0 to t=15 with step size of 0.05 [sec]
mas.run(0, 15, 0.05)
```

Modeling AMAIS: Class Diagram



Interaction of agents with the knowledge base

To access the knowledge base, software agents use both clear and fuzzy SQL queries. An example of a fuzzy query involving two linguistic variables (Performance, Price) with piecewise linear membership functions, as well as the result of its execution after accessing the table, is shown in the figure.



ID	Denomination	Price	Performance
1	N1	47	3,2
2	N2	25	1,5
3	N3	26	3,9
4	N4	31	4,8
5	N5	23	2,5

SELECT * from TableU_1 where (Price = "low" AND Performance = "high")

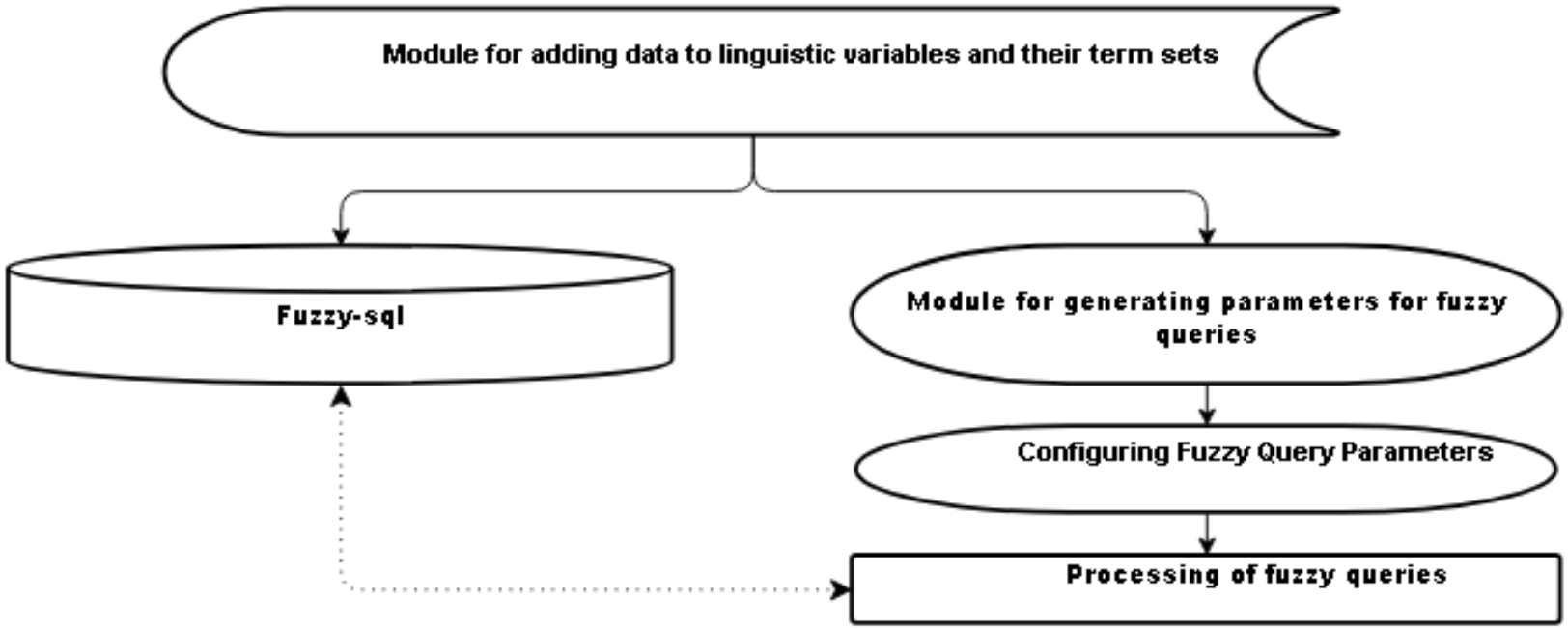
ID	Denomination	Price	Performance	FP
3	N3	26	39	0,8
4	N4	31	48	0,7

a term-set of values linguistic variable (LV) “Performance": "low", "medium", "high“

a term-set of values LV “Price": "low", "medium", "above average“

As a result of a vague query for selecting objects “with low price and high performance”, we have two records that cannot be obtained using clear SQL queries.

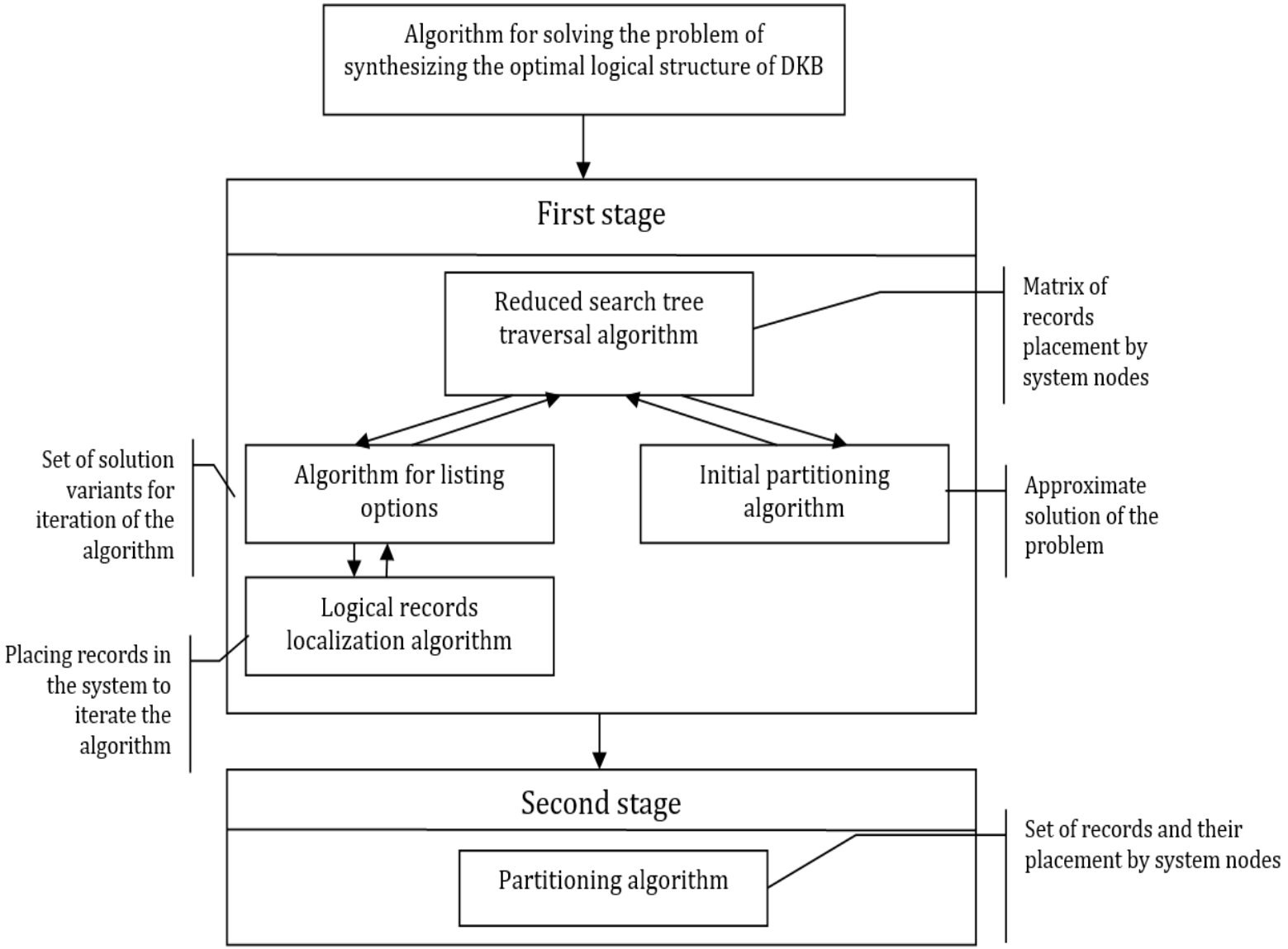
Fuzzy query processing subsystem



Metric	With indexing	Without indexing
Data access operation	Scanning a cluster index	Key Search
Request cost	12.013	0.089
Input/output operation time	8.874	0.003
CPU time (ms)	219	16
Elapsed time (ms)	195	63

Query execution speed

Synthesis of the logical structure of the knowledge base



Knowledge base optimization

The mathematical formulation of the problem and is of the form

$$\min_{\{x_{it}\, y_{tr}\, y_{jm}\}} \sum_{k=1}^{K_0} \sum_{p=1}^{P_0} \xi_{kp}^{Q} \varphi_{kp}^{Q} \{\sum_{t=1}^{T} w_{st}^{n} [\sum_{m=1}^{M} (1 - Z_{sr}^{m} \delta_{pr})(t^{acc} + t_{rm}^{transf}) + \sum_{r=1}^{R} Z_{sr} (t^{task} + t_{rm}^{ser} + t_{rm}^{transf})]\} \quad (1)$$

In here t^{task} – **average duration of forming one request as part of a transaction;**

t_{rm}^{ser} – **average duration of route selection, establishment of logical connections between the client node (r) and the server node (m);**

t_{rm}^{transf} – **average duration of data partition (logical record) transfer from client node (r) to server node (m) along the optimal path;**

t^{acc} – **average time of access to the DKB and record search;**

ξ_{kp}^{Q} – **frequency of use of a request (p) by a transaction (k);**

φ_{kp}^{Q} – **binary sign of the use of a request (p) by a transaction (k).**

Criteria for the effectiveness of the synthesis task

Table 1. Table captions should be placed above the tables. Criteria for the effectiveness of the synthesis task

Variable	Values
X_{it}	Determines the allocation of logical entry (i) of the RBZ to partition (t): $X_{it} = 1$, if the entry made it into the section; $X_{it} = 0$, otherwise
Y_{tr}	Specifies the placement of partition (t) on the server node (r) of the distributed system: $Y_{tr} = 1$, if the partition is hosted on the system node server
W_{st}^{n}	Determines whether a transaction (s) consisting of (n) queries updates a partition (t): $W_{st}^{n} = 1$, if the section has been updated
Z_{sr}	The variable P_{sr} defines the set of nodes (r) of DKB servers fixed by transaction (s): $$P_{sr} = \sum_{t=1}^{T} \sum_{i=1}^{I} w_{st}^{n} x_{it} y_{tr}$$ $Z_{sr} = 1$, with $P_{sr} \geq 1$
$Z^{m}{}_{sr}$	Показатель P_{sm} defines the set of (m) DKB server nodes containing transaction (s) data of partition (j): $$P_{sm} = \sum_{j=1}^{J} w_{sj}^{n} y_{jm}$$ $Z_{sr} = 1$, with $P_{sr} \geq 1$

Thanks for your attention