SIMULATION MODELING ENVIRONMENT FOR COMPLEX DATA DRIVEN DISTRIBUTED SYSTEMS DESIGN

Peter Panfilov, Sergey Salibekyan & Yury Leokhin

Abstract

The methodology is presented for the data-driven distributed application development which is based on the proprietary object-attribute model of the computational process and data organization, which allows for modeling the parallel processes according to the dataflow principle and to synthesize and modify the data structures directly in the simulation process. We propose a new formalization approach for specification and analysis of dataflow applications that combines the proprietary OA-approach to describe the data synthesis and modifications, the KPN to describe the parallel dataflow computational process, and the theory of automatons to describe the operation of a separate FUs in application. The proposed methodology is realized as an object-attribute programming and simulation environment which is based on and build upon a proprietary OA-programming language.

Keywords: distributed systems; dataflow; object-attribute model; programming and simulation environment.

1. Introduction

In this paper, we discuss new approach to simulation modeling of distributed dataflow computer systems and applications targeted at processing complex structured data such as those found in computer-aided design/manufacturing systems (CAD/CAM), expert and decision support systems (DSS), artificial intelligence systems (AI), and search engines. The contemporary computer systems do not cope well with this problem for different reasons. First, the von Neumann architecture which is based on control flow paradigm that dominates the computer world these days is not flexible enough to construct the complex distributed computational processes. Second, there is a lack of adequate formal models for specifying and performing different kinds of analyses for such systems. We here present a survey of basic models which can pretend to play a significant role in this field.

The automaton model (Final State Machine, FSM) is more suitable for modeling sequential processes. But there are attempts to adapt it to concurrent processes and parallel computations by creating a system of parallel operating automatons which exchange the state numbers or react to events generated by other automatons contained in the computer system [1]. But synchronization of such a system is very complicated because the computer designer must do this manually. The state of affairs with the complexly structured data processing is even worse.
The Petri nets are mainly intended for searching the deadlock conditions in the modeled system (also in parallel systems) but is completely unsuitable for modeling the complexly structured data processing. Moreover, they are inconvenient for modeling the parallel processes because, according to the opinion of Carl Hewitt, Professor of Massachusetts Technology Institute, USA, they have the following significant drawbacks [2]:

- the Petri nets are restrictive because they model the data flow control rather than the data flow itself
- they are too complex to describe simultaneous actions in the computational process
- the “physical” interpretation of transitions in the Petri nets is rather dubious

Different process algebras (π-calculus, Communicating Sequential Processes (CSP), Calculus of Communicating Systems (CCS), etc.) are suitable for modeling the parallel computations (they describe the computational threads and the data exchange between them), but they cannot model the complex data processing.

The frame (or object-oriented) model [3] is well suited for simulation of complexly structured data. This model permits describing highly abstract complex structures and combines the data and the program (the object behavior) in a single information structure. But the information structures are rather restrictive, namely, such models can operate only with a tree-like structure of the data. The OO-paradigm meets difficulties in the simulation of parallel process, because it cannot be used these purposes. The drawbacks of the object-oriented approach are described in [4].

The agent model where the system is represented as a set of agents, i.e., autonomous mutually interacting objects with their own behavior, is ideal for modeling the parallel processes. The agents can be organized in structures (for example, as graphs of agents). It should be also noted that the agent system does not require any centralized control and is very flexible, which permits modeling the self-organizing systems. But several authors (for example, [5]) note that nowadays there are no efficient agent models available for understanding and practical use of architectures. The development of general methodological approach for constructing such a class of models including all stages of development (statement of the problem, its formalization, the simulation program realization, and experiments) is still in progress. There are still no conventional formal models in this field.

A very perspective research area in the simulation modeling is the dataflow (control of computations by using the data flow), because it is ideal for describing the parallel processes. The processes are then synchronized according to the data (i.e., the synchronization occurs as if by itself), which significantly simplifies the modeling of parallel computer systems. The system does not require any centralized control and hence permits modeling self-organizing systems. The dataflow system is a graph whose nodes are executive units exchanging messages which form a token along the graph arcs [6]. A token is a set of data together with some ordering information (the message receiver address, data type, etc.). The process in such a system an avalanching process, i.e., the messages arrive at executive units; after an executive unit obtains all necessary data, it activates and begins to process them; then the obtained data formed as a token arrive at other executive units; etc.

At present, a method for processing the complexly structured data, which is called the i-structure, is most frequently used in the dataflow-paradigm [6]. The i-structure is a data structure where each field has a data flag. The data in the field is written only once, and hence no modification of the data, not to mention the data structure, can be there.

A commonly accepted formal model in the field of dataflow is the Kahn process network (KPN) which is a set of calculators (or handlers) exchanging tokens through data channels equipped with the processing queue (if an executive unit is processing the preceding token, then the arriving token in placed in the queue until the unit is free). The KPN formalism is ideal for modeling parallel systems, but it does not permit modeling a unique data structure processed by several simultaneously operating parallel executive units.

The discrete event simulation (DES) [7] describes only methods for modeling parallel processes and does not pay attention to the technology of simulation of the data structure processing.

Therefore, the goal in our investigation is to construct (or choose) the simulation method, to develop the mathematical model, and to create the simulation environment for such a class of systems.

2. Simulation modeling methodology in OA-basis

An object-attribute (OA) approach for organizing the computational process and data structures [8, 9] belongs to the dataflow class [6] and underlies the developed methodology because it permits modeling both the parallel processes and any modifications of complex data structures.

Comprehensive facilities for modeling the complex data structures are provided by the OA-graph, which allows one to describe structures of any complexity and topology and to modify them. In fact, the OA-graph is a semantic net whose nodes can denote both physical and abstract objects; the nodes are described by information capsules (IC) which are sets (or sequences) of information pairs (IP). The OA-graph can easily be synthesized or modified by adding or deleting an IP to IC, i.e., the OA-graph is similar to a toy construction set, and any data modification does not violate the data structure integrity. The OA-graph is processed by functional units (FU) which exchange information in the computational process; the information is organized as tokens; the tokens have the simplest form in the OA-system, i.e., they consist of two fields, the attribute (data identifier) and the load (data or reference-pointer).

The object models described by OA-graphs can be endowed with behavior (similarly to the object-oriented approach or OOA), because the OA-graph nodes can also be subroutines (similarly to the OOA methods). An OA-program is a
sequence (or a set) of IP (or millicommands) transferred to FU; it can be placed in IC and built in the OA-graph. At any moment, an IP sequence from such an IC can be transferred to the corresponding FU to be processed. The IP loads can also contain the FU pointer to the corresponding OA-program to be performed.

Moreover, the FU themselves can also be structured, because their context (the set of internal registers determining the FU state) can contain pointers to the contexts of other FU (the context is the memory region containing the content of the FU internal registers). For example, to organize the FU as a grid structure, it is necessary to supplement the context of each FU with four registers containing references to the contexts of the neighboring FU (on the right, on the left, above, and below). The FU then use these references to exchange data with their neighbors. Thus, for example, the physical processes can be modeled by the net approximation method, where the considered physical region is divided into several regions by using a grid and the physical region characteristics are computed at the nodes of this grid (each node is associated with its own FU). In such computations, the parameters of the current point on the grid depend on the parameters of the neighboring nodes. The FU structures can be reconstructed in the process of modeling; for this, it is necessary to change the addresses of the computation result receivers in the FU contexts. Thus, both the structure of connections between the FU and the data structure can easily be modified in the model constructed according to the OA-principles.

It should be noted that the proposed approach is versatile. First, it ensures the simulation of structures of any topology, namely, the OA-graph structure can be regular (for example, hypercube, tree) or irregular (net). Second, it is possible to model the self-organizing systems and systems synthesizing and modifying their structures dynamically, because there is no centralized control in the model (the computational process is controlled by a set of independently operating FU). For example, this holds for the semantic analysis of a natural language in the case where the ontological base describing the text meaning is constructed in the analysis of the text; or for the collective behavior of some intelligent objects (for example, a collaborated mobile robots); or for Bionic Assembly System [10].

3. Formalization basics of an OA-model

It was required to formalize the developed OA-approach. The formal model must equally effectively describe the parallel dataflow computational process and the process of data synthesis and modification. To satisfy these requirements, we combined the OA-approach (the notions of IP, IC, and OA-graph) to describe the data synthesis and modifications, the KPN to describe the parallel dataflow computational process, and the theory of automatons to describe the operation of a separate FU.

3.1. Description of data structure

The data in the OA-system are described by IP, IC, and OA-graphs. We now formalize these notions and construct the so-called OA-algebra which allows us to describe such constructions.

We begin with the notion of information pair (IP). An IP is the dyad \( \{a,l\} \), where \( a \in A \) is the set of attributes and \( l \in L \) is the set of loads. The attribute \( (A \subseteq \mathbb{N}) \), where \( \mathbb{N} \) is the set of positive integers) is a universal identifier of the load used by the FU to recognize it. The load \( l \in L = \{\text{nil} \cup \Omega \cup \Theta \cup S \cup A\} \), where \( L \) is the data set in the load, \( \mathbb{N} \) is the set of rational numbers, the symbol “nil” denotes the empty load, \( \Omega \) is the set of global memory addresses (numbers), \( \Theta \) is the set of FU addresses (numbers), and \( S \in \Sigma^* \) is the set of symbol chains in the alphabet \( \Sigma \).

All IP can be divided into two classes, namely, the information IP (used to describe some objects) and the millicommands (used to exchange information between OA-automatons).

We define IC by the definition of “IP chain”. We shall say that \( \eta \) is an IP chain, if the following conditions are satisfied:

1. if \( \varsigma \) is an empty IP chain (does not contain any IP), then \( \varsigma \) is an IP chain;
2. if \( \gamma \) is an IP chain and \( \iota \) is an IP, then \( \gamma \iota \) (or \( \gamma \cup \iota \)) is an IP chain (where \( \cup \) is the symbol of union of IP chains);
3. \( \eta \) is an IP chain if and only if it is conditions (1) and (2) are satisfied.

We use \( |\eta| \) to denote the cardinality (number of IP) of the IP chain \( \eta \).

To modify IP, we introduce the operation of concatenation (union) of IP chains in the OA-algebra: if \( \alpha \) and \( \beta \) are IP chains, then the IP chain \( \alpha \beta \) (or \( \alpha \cup \beta \)) is called the concatenation (chaining, union) of IP chains \( \alpha \) and \( \beta \): \( \alpha \beta = \alpha \cup \beta \), where \( \omega \) is an empty IP chain. We also introduce the operation of deleting the last or the first IP from a chain, of deleting an IP with a certain index, of introducing an IP chain in another chain after an IP with a certain index, and some other operation modifying IP chains.

To operate with several IP chains in the model, we introduce the notion of bunch of IP chains \( (W) \) which is an interrelated data structure consisting of IP chains. The cells of the bunch \( W \) can contain IPs \( W = [w_1, w_2, \ldots, w_{|W|}] \), where \( |W| \) is the length of the bunch of symbol chains, \( w_1, w_2. \ldots w_{|W|} \). To index a specific IP in the chain of IP chains, we can use the double index \( W_{ij} \) denoting the IP contained in the \( j \)-th IP on the \( i \)-th IP chain in \( W \). In our model, a bunch of IP chains can emulate the common main memory of the computer system, where the data to be processed are stored; this memory can be accessed by several computing devices at once.
The OA-graph in our model is a set of IP chains united by references located in IP loads (the references belong to the set of addresses of global memory $\Omega$) (Fig. 1). For convenience, the chains are united in a bunch of chains so as to operate with certain chains according to their indices.

![Fig 1. An object-attribute graph](image)

### 3.2. Modelling the computational process

The model used to describe the process is based on the KPN, where the messages transferred between the nodes are symbols of a certain alphabet. A message in our model is an IP (millicommand). The FU uses the attribute to identify the obtained data stored in the millicommand load and appropriately processes them. If the FU cannot process the obtained millicommand, it is placed in the queue.

The FU operation can be described by an “improved” automaton model. The improvement means that the automaton deals not with the input flow of symbols but with the flow of millicommands. In certain cases, the use of OA-automata with discrete states is insufficient, because the OA-automata must store the data arriving in the millicommand loads (recall that $L = \{\mathbb{R}, \mathbb{N}, \mathbb{S}, \mathbb{O}, \mathbb{E}\}$ and the current state can also be described as a continuous multidimensional space). Therefore, we had to use the formal apparatus of hybrid systems which describe the automata with continuous space of states. So we first give the formal definition of OA-automaton.

The **OA-automaton context** is a set of abstract entities of any nature (scalar or vector variables, countable or continuous sets, etc.). If each entity contained in the OA-automaton context is represented as a measurement in a multidimensional space, then the automaton state $K$ is a point in this space. The OA-automaton context must necessarily contain input and output queues of millicommands which are used to exchange information between OA-automata. A variation in the OA-automaton state is called its activation. An activation may occur only if one of the input millicommand chains contains a millicommand (i.e., there is a millicommand to be processed). Such a transition is determined by a function $F$ contained in the OA-automaton context $K'=F(K)$, where $F$ is the function of the OA-automaton transition from the current state $K$ to a new state $K'$.

For example, the function $F$ of an OA-automaton with discrete states can be defined by the transition predicate vector $\vec{P}$ and the transition function vector ($\vec{F}$). In this case, the vector dimensions must coincide, i.e., $|\vec{P}| = |\vec{F}|$. The arguments of the predicate $P_i$ and the function of transition to a new state $F_i$, where $i=1,2,...,|\vec{P}|$, are all the parameters contained in the context $K$. Here $F_i$ is a vector of functions, where each function returns the value of one of the context parameters $F_i = [F_{i1}, F_{i2}, ..., F_{ik}]$, $K_j = F_j(K)$, where $j=1,...,|K|$.

The OA-automaton transition from state $K$ to a new state $K'$ occurs as follows: the predicates in the predicate vector are considered in succession and if $\exists i: P_i(K) = true$, then $K' = F_i(K)$ (i.e., $K_j = F_j(K)$, where $i=1,2,...,|\vec{F}|, j=1,2,...,|K|$) and the search is terminated. In the case where none of the predicates $P_i$ is true, the OA-automaton context remains unchanged. The function $F$ is contained in the OA-automaton context and can be changed after the OA-automaton transition to a new state. Thus, the automaton operation algorithm can be changed directly in the computational process.
The OA-automata can be divided into several types each of which realizes a certain functional set and has its own individual context. But the context of an OA-automaton of any type necessarily contains the input and output queues of millicommands. So the context variation function $F$ can be activated only if there is a millicommand at least in one of the input queues of millicommands (after activation of the function $F$, the millicommand at the head of the millicommand queue is deleted). When changing its context, an OA-automaton can place the information at the end of its output millicommand queues and thus transfer the data to other OA-automata for which this queue is an input queue. In this case, one millicommand queue can be contained in the context of one OA-automaton as an output queue and in the context of another OA-automaton as an input queue. The context of an OA-automaton of any type necessarily contains the common main memory $W$ (an IP chain) which stores the OA-graphs for the common processing by OA-automata. Thus, the OA-automata can operate with the data located in this memory (this ensures the OA-automata operation with the common memory). All OA-automata are designed on the basis of the base automaton which has the minimal set of context fields to ensure the operation of the OA-system (OA-network):

$$AB = \{ A, L, MkIn, MkOut, W, F \},$$

where $A$ is the alphabet of attributes, $L$ is the data set in the IP load, $MkIn$ is the queue of input millicommands, $MkOut$ is the queue of output millicommands (an OA-automaton can add an Mk chain to the end of the queue), $W$ is a global memory (bunch of IP chains), $F$ is the function of the OA-automaton context change (each type of OA-automaton is associated with its own context change rules).

$F$ enters the context and hence can be changed in the transition of the OA-automaton to another state. The possibility of dynamic reconfiguration (reprogramming) of an OA-automaton in the course of the computational process is thus realized.

A new type of automaton can be obtained by adding new elements to the context of the base automaton.

The set of OA-automata forms an OA-automaton network. Some context elements can simultaneously be contained in several OA-automata. For example, one millicommand queue can be an input millicommand chain for one automaton and an output chain for other automata. So, one automaton can place a millicommand in a queue where it is taken to be processed by another OA-automaton. In Fig. 3, we outline the OA-network where the data exchanged is realized through a special FU called «Bus». In such a system, all input millicommand queues are input queues for the FU Bus, and the FU output millicommand queues are input queues for the FU Bus. The FU Bus obtains a millicommand into its output queue from FUs, processes it, determines the millicommand target, and places it in the input queue of the corresponding OA-automaton-target.

Fig. 2. Graphic representation of the OA-automaton context change function

- 0395 -
A sequence of OA-automaton activations in the OA-network is considered as an implementation of the OA-network. Only one OA-automaton can be activated at a moment of the model time. The initial data for the OA-network are formed as millicommands and are placed in the input millicommand chains of the corresponding OA-automata. The OA-network implementation is finished when all input millicommand queues are empty or when a different scenario is used. The states of OA-automata and the millicommands in the output millicommand queues form the simulation result.

4. Language for the OA-model description

A special programming language (OA-language) was developed for practical applications of the proposed OA-modeling and simulation methodology [8]. This language allows one to create virtual FU (or OA-automata), initialize them, input the initial data in the OA-model, and output the simulation results. The OA-language is also used to construct OA-graphs which are then modified by a set of OA-automata (FU).

4.1. OA-language syntax

The Object-Attribute language has a very simple syntax (see Table 1). An IP is denoted by the symbol «=», the attribute mnemonics is placed to the left of this symbol (each index of the attribute is associated with a mnemonics), and the load notation is placed to the right. The load can be a constant (integer or fractional number, Boolean value or row), a variable, or a reference. A capsule is denoted by curly brackets. If an IC is given as an IP load, then it can be written in curly brackets immediately after the symbol «=». A new FU is created by the millicommand NewFU, and an IC with the FU parameters is indicated in its load, namely, the FU type and its mnemonics. In the environment, each FU has its universal name (mnemonics) which is used by the programmer to address it. The mnemonics of a millicommand
transferred by an FU is written after the FU mnemonics and is separated by point. An OA-graph in the «list» topology is denoted by the symbol «>».

<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>InformationPair (IP)</td>
<td>Attribute=Load</td>
<td>FUType=FUUIntALU</td>
</tr>
<tr>
<td>Millicommand</td>
<td>FUName.DataAttribute=Load</td>
<td>ALU.Set=0</td>
</tr>
<tr>
<td>Capsule</td>
<td>[CapsuleName][{IP1 IP2 …}]</td>
<td>[ALU.Set=0 ALU.Add=5]</td>
</tr>
<tr>
<td>Capsule reference</td>
<td>{Attribute=[CapsuleName]{IP1 IP2 }}</td>
<td>ALU.EqProgSet={Console.LnOut=&quot;Zero&quot;}</td>
</tr>
<tr>
<td>Declaration of a New FU</td>
<td>NewFU{Mnemo=FUName F&gt;Type=FUType}</td>
<td>NewFU{Mnemo=&quot;ALU&quot; FType=FUIntALU}</td>
</tr>
</tbody>
</table>
| Comments                 | \
* … *=\                         | \
* to the end of the row
* comments *=\                       |
| Declaration of Variable-Load | FUName.DataAttribute=VariableName (Load) | ALU_Fib2.Set=temp(0) |
| Load with Reference to Variable | FUName.DataAttribute=VariableName | ALU_Fib.Pop=temp
ALU_Fib2.Set=temp(0)!
| List                     | >[CapsuleName]{...}           |                                   |
|                          | >[CapsuleName]{...}           |                                   |

Table 1. The Main constructions of the OA-programming language

4.2. Programming in OA-language

For example, in the OA-language an FU is constructed, initialized, and controlled as follows:

NewFU={Mnemo="ALU" FType=FUIntAlu} // construct an FU with name «ALU»
ALU.Set=1 // Set the accumulator value 1
ALU.Mul=10 // Multiply the accumulator by 10
ALU.PopMk=Console.LnOut // Output the obtained value to the console.
// Console is an FU that outputs the result to the console.

An example of the OA-graph description (in this case, we describe the part of the dictionary which is used in the system of semantic analysis of the natural language in the NLP application):

Vocabulary.Set=
>:{Mnemo=’coach”
  Fork=
    >:{PartOfSpeach=Noun SemProp={ Contiguity =Sport Contiguity=Instructor}}
    >:{PartOfSpeach=Noun SemProp={ Contiguity =Transport Contiguity=Railway}}
    >:{PartOfSpeach=Noun SemProp={ Contiguity =Transport Contiguity=Horse}}
  >:{PartOfSpeach=Verb SemProp={ Contiguity =Transport Contiguity= Horse Contiguity=Moving}}}

In this case, we describe the multi-valued word «coach» which can mean, first the trainer of a sport team; second, a railway car; third, a horse-drawn carriage; fourth, the verb «go by a carriage». The attribute Fork precisely means that the IP load contains a reference to the list of meanings of a single word. SemProp is the attribute of the pointer to the semantic characteristics (properties) of the word. Contiguity is an associative relation (each word has individual associative relations). PartOfSpeach denotes the part of speech to which this word belongs. A special FU is designed to process OA-graphs of type «list»; in our example, such an FU has name Vocabulary, and the FU millicommand Set is used to transfer the reference to the OA-graph.
5. Programming and simulation environment for the OA-applications

We have developed and tested the methodology of simulation modeling of data-driven applications that follow and implement the dataflow paradigm of computing. This methodology is realized as an object-attribute programming and simulation environment which is based on and build upon a proprietary OA-programming language as it presented in previous Section. The OA-environment consists of the following parts.

- Software modules realizing the FU operation logic (OA-platform). Each FU type is a subprogram in the imperative programming language with standard interface. Such an approach is very similar to the actor programming, but the actors have different interfaces which does not improve the unification of the modeling system. The interface of the FU subprogram has three fields: the reference to the memory region containing the FU context; the millicommand index; and the reference to the millicommand load.
- Compiler of the OA-language of programming. It should be noted that the compiler is realized on the basis of the OA-architecture and is written in the OA-language (we used the technology of compiler untwining).
- Console of OA-program input and the console of the program operation result output. It is also possible to create additional input and output consoles. There also exist program modules responsible for the output of the OA-graph and of different graphs.
- Tool panel (on the left in Fig. 5), simplifies the process of programming (list of mnemonics of attributes, variables, pointers, list of FUs created in the environment).
- Tools ensuing the simulation modeling according to the Discrete Events System (DES) principle [7].

Fig. 4. Interface of OA-environment for data-driven application development and simulation

Our OA-programming and simulation environment is in development phase. At present, more than 80 types of FU have been realized in the OA-environment. There are several types of FU ensuring the OA-language compilation, information processing, input and output of information, and OA-graph processing, and there are FU designed to solve some application-specific problems. Because the OA-approach for organizing the computational process is very flexible, FU of new types can be added to the OA-environment without violation of the entire environment integrity. Any modification of the program implementing the FU operation logic does not also violate the environment integrity, i.e., the FU modification consists in adding new millicommands to the program while the old millicommands preserve their functionality. The refurbishment (which is practically inevitable when rather complicated OO-programs are designed) can be avoided because of the universal interface of the program implementing the FU operation logic. It allows one, without any aftereffects, to change the number of arguments of the commands realized by FU and to supplement FU with new operation modes without apprehensions to violate the system integrity.

6. Conclusion

The methodology of simulation modeling and its formal model were developed. The methodology is based on the OA-principle of the computational process organization and data organization, which allows us to model the parallel processes according to the dataflow principle and to synthesize and modify the data structures directly in the simulation process. To develop the formal model, we analyzed the existing methods and proposed a model based on a combination
of the following three methods: the OA-approach for organizing the computational process and data structures, the hybrid automaton model, and the KPN. As a result of our investigations, we constructed the environment of OA-programming and simulation which permits modeling systems with complex dynamically variable structure. The proposed approach allows one to realize models of any level of detail, from conceptual (ESL) to one-to-one models.

The developed methodology permits simulation on multiprocessor and multicomputer complexes. The point is that the FU are autonomous devices «communicating» with other virtual FU (VFU) only through messages, i.e., the VFU are not in the universal address space and can therefore be set in operation on different computational devices united by communication lines (the messages between the VFU are transmitted through these channels). The CS where the simulation is implemented can also be heterogeneous (i.e., can consist of computational nodes of different architecture). Such possibilities can significantly accelerate the simulation process and make the CS much cheaper (the CS consists of many low-power computational nodes, and this is much cheaper than one high-performance CS).

The OA-model design does not require highly qualified programmers. For example, the program can be realized in the imperative language, and one need not insert special language structures in the text of the program (as in the case of the actor programming). The OA-programming language used to describe the FU interaction has extremely simple syntax and minimal time is required to get acquainted with it. Because of the dataflow-paradigm, the programmer need not take care of synchronization of the modeled parallel processes, i.e., the entire synchronization is automatic over the data.

The proposed OA-programming and simulation environment can also be used for modeling and study the system of semantic analysis of a natural language (Natural Language Processing, NLP). The simulation model of such a system can synthesize a semantic net describing a certain ontology from a text in a natural language. The model also takes the polysemanticism (multiple meaning) of lexemes and syntax structures into account.

The process of semantic net synthesis begins with formation of a list of lexeme interpretations. Then this list is transformed into a semantic net in several stages. In the English language, more than 20 stages have been separated (but it may happen that the number of stages will increase as a result of further studies). At each stage, two or more lexemes are united in a single semantic structure and so, step by step, the list of lexemes becomes a semantic net. At present, an FU specializing in transformation of the list of initial lexemes in the semantic net (OA-graph) is integrated in the OA-environment and some experimental bases of knowledge necessary to analyze the Russian and English languages are realized. The NLP-model operates with a restricted dictionary and processes a certain set of rules describing the identified language. The principle of lexeme semantic matching was developed to delete the unnecessary lexeme interpretations and text fragments, which permits omitting the lexeme interpretations inconsistent with the context. For example, the obtained semantic net can be used to seek information, namely, the user request in the natural language is transformed in the semantic graph-request, and it is further determined whether the graph-request is a subgraph of the text graph. If the answer is affirmative, then the search result is positive; otherwise, the result is negative.

The developed environment, which was used to model a special system of matrix multiplication, a system for modeling the computational processes by the net approximation method, and a system for modeling the semantic analysis of a natural language, demonstrated its efficiency.

7. References


