Model Synthesis and Model-Oriented Programming

Yury I. Brodsky

Institution of Russian Academy of Sciences Dorodnicyn Computing Centre of RAS,
Vavilov st. 40, 119333 Moscow, Russia

yury_brodsky@mail.ru

Abstract. *The work is devoted to the description and simulation of complex systems, about which it is well known of what components they are made, what those components are able to do, what rules of interaction they obey. The challenging problem of modeling is to reproduce the behavior and to evaluate the capabilities of such a system as a whole. A new approach to the design and implementation of computer simulation models of complex multicomponent systems is introduced. It differs from the object-oriented approach. The central concept of this approach and at the same time, the basic building block for the construction of any more complex structures is the concept of the model-component. Model-component endowed with a more complicated structure than, for example, the object in the object-oriented analysis. This structure provides to the model-component an independent behavior - the ability of a standard way to respond to standard requests of its internal and external environment. At the same time, the computer implementation of model-component's behavior is invariant with respect to the integration of models-components into complexes, which allows firstly to build a fractal model of any complexity and secondly to implement a computational process of such structures uniformly - by a single program. In addition, the proposed paradigm of the multi-component simulations allows to exclude imperative programming code and to generate code with a high degree of parallelism.*

Keywords

Simulation, Complex Multi-Component Systems, Model Synthesis, Model-Oriented Programming, Parallel and Distributed Computing.

1 Introduction

In terms of non-formal understanding of what is a complex multicomponent system, very close to the author are the following three statements by prominent specialist in the field of complex systems and their simulation, correspondent member of the USSR Academy of Sciences, N.P. Buslenko, derived from his works [1, 2]: "A complex system is a complex object whose parts can be regarded as systems, logically combined into a single entity, in accordance with certain principles or defined set of relations", "In every moment of the time every item in a complex system is in one of its possible states; it passes from one state to another under the influence of external and internal factors", "To construct a synthesis of complex system's behavior, we must give to its components the opportunity to fully express themselves". In some sense, all that is proposed in this paper is one of the possible implementations of the above statements.

Fully agreeing on an intuitive level that the complex system itself may consist of complex systems, we do not undertake to give a rigorous definition of this concept. However, the definition of a model of a complex system will be put on quite formally, as a species of structure in the sense of N. Bourbaki [3]. From this point of view, a complex system – is what can be adequately enough represented by a model of a complex system. A species of structure is a development of the concept of a set. A base set is supplied with a structure of some species - a certain type of relations between its elements, and depending on this relationship, the set can become, for example, a group, or a lattice, or a vector space, or in our case – a simulation model of a complex system. The mathematical object, for example a specific linear space, is an instance of the structure of the respective species.

The structuralism method [4], which ideologically goes back to the F. Klein's Erlangen program [5], and turned out to be quite popular and productive in the twentieth century including the humanities, offers to consider further various mappings of base sets and look for the invariants of these mappings. For example, the morphisms - the mappings of base sets that keep the species of structure. The school of correspondent member of RAS Yu.N. Pavlovsky at CC RAS, in recent decades developed the geometric theory of decomposition [6-9], where with the help of morphisms they are trying to find simple mathematical representations of various mathematical objects - their reductions and decompositions.

In this work, devoted to the synthesis of a complex system model from models of its component, we are mostly interested in the possibility of extending the species of structure on consolidate base sets of mathematical objects, endowed with this species of structure. An invariant with respect to the integration of base sets of simulation models is the method of simulation calculations organization, proposed below.

Model synthesis as an alternative to the object-oriented analysis way of describing and synthesis of simulation models of complex multi-component systems, was developed in the CC RAS Department of Simulation Systems since the late 80's. Its main ideas and methods are described in the works [10-14], however, the term "model synthesis" was firstly proposed only in [10]. In the base of the model synthesis lays the concept of the model-component.

From the point of view of software systems constructing, the model-component is similar to the object of object-oriented analysis, but in addition to the characteristics, equipped with not only the methods that can do something useful, but with some equivalent of an operating system which is always ready to give standard answers to standard requests from the internal and external environment of the model.

From a formal point of view, the model-component is a mathematical object, which has as the base set an aggregate of sets of internal and external characteristics of the model, methods (what the model is able to do) and events (what the model should be able to respond.) At the base set is introduced a species of structure "model-component", which has two remarkable features:

1. The species of structure "model-component" provides a standard and unambiguous way to organize the computational process of simulation for all objects provided with this species of structure. This means that you can create a universal program that can run on any implementation of a simulation model, if that is a mathematical object with the species of structure "model-component".
2. Generally speaking, if we consider two arbitrary mathematical object equipped with the same species of structure (for example, the species of structure of an abstract group), the enhancement of this structure on the union of their base sets is not always possible. However, for the species of structure "model-component", such a spread of the common species of structure of the components onto the association of their base sets is possibly (if a subset of the characteristics of their basic sets have not pairwise intersections), or possibly with some justification (for example, by refilling the original number of components with some additional components, with the same species of structure).

The second property lets us form models-complexes from the models-components by combining their base sets and after the spread of the species of structure on the union of base sets, the models-complexes become mathematical objects of the same species of structure "model-component", and therefore, again can be combined in models-complexes. The first property allows us not to despair of the computational complexity of the process, which is obtained as a result of such overcomplicated fractal associations.

For software implementation of complex multi-component model a model-oriented programming paradigm is proposed, where the unit of software design is a model-component – a more aggregated construction than the object in the object-oriented analysis.

It will be shown that the model-oriented programming allows to eliminate the imperative programming [14] - the most difficult for both the development and the debugging. Furthermore, the resulting executable code has a high degree of parallelism and the degree of the code parallelism increases with the model complexity increasing. This fact may offer a scope for applications of model-oriented programming in high-performance computing systems, including for tasks not related to the simulation, but with a multi-component organization.

2 Related works

There are many tools to build models of complex multicomponent systems, including commercially successful, for example, AnyLogic by XJ Technologies. A detailed review can be found in [11]. However, all known to the author, offering their own ways and the concepts of building models, newer try to put the question of why these methods are as follows, according to the principle "victors are not judged". We hope that this work in addition to the recommendations how to build a synthesis of models of complex systems will touch upon the question of why it should be done that way.

The primary means of software implementation models of complex systems are object-oriented programming languages. In our opinion – one of their drawbacks is that they propose an imperative style of programming, where it is easy to make bugs and difficult to debug. Also in our opinion these languages have too low level of encapsulation - creating a software system, we have to call a number of different methods from different classes and therefore all the time are to keep the mind on their signatures. Detailed comparison between object- and proposed in this paper model-oriented programming will be held later in a separate section.

Finally, the complexity of the simulated systems causes a natural desire to use the facilities of high-performance computing systems. Here, however, we don't know system solutions for parallelization of simulation computations. This paper proposes a way of parallelization of simulation computations.

3 The Species of Structure “Model-Component”

Let's try to define formally a mathematical object that represents an elementary simulation model of a complex system.

$$
\begin{aligned}
&\textit{Model-Component} = < X, M, E, \{M_j\}_{j=1}^{N}, \{E_j\}_{j=1}^{N};\\
&x \subset X,\ \{m_{j,real} \subset M_j \times M\}_{j=1}^{N},\ \{e_{j,real} \subset E_j \times E\}_{j=1}^{N},\ \{m_{j,in} \subset M_j \times \beta(X)\}_{j=1}^{N},\\
&\{m_{j,out} \subset M_j \times \beta(X)\}_{j=1}^{N},\ \{e_{j,in} \subset E_j \times \beta(X)\}_{j=1}^{N},\ \{m_j^0 \subset M_j\}_{j=1}^{N},\ \{sw_j \subset E_j \times M_j \times M_j\}_{j=1}^{N},\\
&\{p_j \subset \beta(M_j) \times \beta(E_j) \times M_j \times \beta(E_j \times M_j \times M)\}_{j=1}^{N};\\
&R_1: \{(\forall m \in M_j)(\exists ! \tilde{m} \in M)(\{m, \tilde{m}\} \in m_{j,real})\}_{j=1}^{N},\ R_2: \{(\forall e \in E_j)(\exists ! \tilde{e} \in E)(\{e, \tilde{e}\} \in e_{j,real})\}_{j=1}^{N},\\
&R_3: \{(\forall m \in M_j)(\exists ! r \in \beta(X))(\{m, r\} \in m_{j,in})\}_{j=1}^{N},\ R_4: \{(\forall m \in M_j)(\exists ! r \in \beta(x))(\{m, r\} \in m_{j,out})\}_{j=1}^{N},\\
&R_5: \{(\forall e \in E_j)(\exists ! r \in \beta(X))(\{e, r\} \in e_{j,in})\}_{j=1}^{N},\\
&R_6: \{((\forall e \in E_j)(\exists ! r \in M_j \times M_j)(\{e, r\} \in sw_j)) \,\&\, ((\{e, r\} \in sw_j, \{\tilde{e}, \tilde{r}\} \in sw_j, r = \tilde{r}) \Rightarrow (e = \tilde{e}))\}_{j=1}^{N},\\
&R_7: \{p_j = \{M_j, E_j, m_j^0, sw_j\}\}_{j=1}^{N},\ R_8: \text{there are no identical elements in the sets } X, M, E,\\
&R_9: \text{organization of simulation calculations axiom}>.
\end{aligned}
\tag{1}
$$

Here N – is the number of processes in the model. Everywhere $\{...\}_{j=1}^{N}$ means that the contents of the braces is repeated through a comma N times, while the index j is replaced by $1,...,N$.

Principal base sets of the species of structure are $X, M, E, \{M_j\}_{j=1}^{N}, \{E_j\}_{j=1}^{N}$. There are no auxiliary base sets. The set X is the set of characteristics of the model, occasionally, we will divide it into two subsets: $X = \{x, a\}$, where x – are the internal characteristics of a model that, in accordance with the closeness hypothesis completely determine its status, and a – its external characteristics, that by the same closeness hypothesis completely defines its interaction with all the rest world. The set of different implementations of methods-elements $M = \{s, f\}$, – elementary skills of our model, among which we will sometimes distinguish two subsets: s – slow, realizing a smooth dependence of the internal characteristics of the model from its internal and external characteristics, if for example, $\dot{x} = F(x, a)$, then, $x(t + \Delta t) \approx x(t) + F(x(t), a(t))\Delta t$, and fast f – which implement the internal model characteristics jumps: $\Delta x = G(x(t), a(t))$. The set of different implementations of methods-events E, associated with the model. Events - are something that our model must respond to. The method-event – is a method that gets as an input a subset of model characteristics, and gives as an output a non-negative number, indicating that the event occurred if the output is zero, or the forecast time-to-event, if the output is positive.

Note that in the sets X, M, E there are no identical elements. For X, this follows from the fact that we are talking about the model, and the principle of not increasing the number of entities beyond the necessity (Occam's Razor) encourages us to avoid unnecessary duplication of the same characteristics as of the phenomenon simulated, either of its links with the outside world. As for M and E, the words "a set of different implementations," emphasizes the uniqueness of the methods in these sets.

We will talk about the processes of the model-component. The processes are what can occur in the model simultaneously. It was said above that N is the number of processes. The formal definition of the process will be done later. Each process p_j has the set of its methods-elements M_j and its many methods-events E_j. This concludes the description of the base sets.

We now introduce the typical characterizations on the base set $X, M, E, \{M_j\}_{j=1}^{N}, \{E_j\}_{j=1}^{N}$ and impose on these characterizations certain requirements – axioms of species of structure.

$$m_{j,real} \subset M_j \times M,\ R_1: (\forall m \in M_j)(\exists ! \tilde{m} \in M)(\{m, \tilde{m}\} \in m_{j,real}). \tag{2}$$

For each process p_j a mapping is defined of its methods-elements M_j into the set of implementations M. Thus, for each $m \in M_j$ there is uniquely determined its implementation $\tilde{m} \in M$.

$$e_{j,real} \subset E_j \times E,\ R_2 : \left(\forall e \in E_j\right)\left(\exists !\tilde{e} \in E\right)\left(\{e,\tilde{e}\} \in e_{j,real}\right). \tag{3}$$

For each process p_j a mapping is defined of its methods-events E_j into the set of implementations E. Thus, for each $e \in E_j$ there is uniquely determined its implementation $\tilde{e} \in E$.

$$\left\{m_{j,in} \subset M_j \times \beta(X)\right\}_{j=1}^{N},\ R_3 : \left\{\left(\forall m \in M_j\right)\left(\exists !r \in \beta(X)\right)\left(\{m,r\} \in m_{j,in}\right)\right\}_{j=1}^{N}. \tag{4}$$

Internal and external characteristics of the model are connected to the input parameters of methods-elements of the j-th process. Inclusion determines what subset of the model characteristics is passed to each of the methods-elements.

$$\left\{m_{j,out} \subset M_j \times \beta(X)\right\}_{j=1}^{N},\ R_4 : \left(\forall m \in M_j\right)\left(\exists !r \in \beta(x)\right)\left(\{m,r\} \in m_{j,out}\right). \tag{5}$$

Output parameters of methods-elements are connected to the internal characteristics of the model.

$$\left\{e_{j,in} \subset E_j \times \beta(X)\right\}_{j=1}^{N},\ R_5 : \left\{\left(\forall e \in E_j\right)\left(\exists !r \in \beta(X)\right)\left(\{e,r\} \in e_{j,in}\right)\right\}_{j=1}^{N}. \tag{6}$$

Internal and external characteristics of the model are connected to the input parameters of methods-events of the j-th process. Inclusion determines what subset of the model characteristics is passed to each of the methods-events.

$$\begin{gathered} sw_j \subset E_j \times M_j \times M_j, \\ R_6 : \left\{\left(\left(\forall e \in E_j\right)\left(\exists !r \in M_j \times M_j\right)\left(\{e,r\} \in sw_j\right)\right) \& \left(\left(\{e,r\} \in sw_j, \{\tilde{e},\tilde{r}\} \in sw_j, r = \tilde{r}\right) \Rightarrow \left(e = \tilde{e}\right)\right)\right\}_{j=1}^{N}. \end{gathered} \tag{7}$$

Switching of methods-elements is defined for each process. If under certain conditions it is possible to switch the process from the execution of the method-element $A \in M_j$, to the execution of $B \in M_j$ (and such a condition, according to the closeness hypothesis, can only be a certain combination of internal and external characteristics of the model $\{x,a\} \in X$, – in the virtual world of the model there are nothing at all but its characteristics) it must be created the unique method-event $e_{AB}(x,a) \in E_j$, to calculate and predict such a switching. It is also possible the event of type $e_{AA}(x,a)$, not switching the method-element $A \in M_j$ to any other, but only interrupt its execution (for example, to synchronize the results of its calculations with the other processes of the model). Thus, the time and the order of the elements switching of each process are determined by what happens inside and outside of the model. The mappings from events to switches are injective for all processes.

$$\left\{p_j \subset \beta(M_j) \times \beta(E_j) \times M_j \times \beta(E_j \times M_j \times M)\right\}_{j=1}^{N},\ R_7 : \left\{p_j = \left\{M_j, E_j, m_j^0, sw_j\right\}\right\}_{j=1}^{N} \tag{8}$$

By relation (8) the processes of the model are determined, j – index of the process, which changes from 1 to N. The sets M_j, E_j – are sets of elements and events of the j-th process; m_j^0 – its initial element; sw_j – elements switching rules. Processes – are what may run in parallel in the model, something like when the operating system of the computer at the same time has several system services running. Each process consists of sequential alternation of a finite number of methods-elements. The alternation of elements in the process is event-driven. Distinguished within the process methods-elements and methods-events, generally speaking, may have the same realizations in M, and, accordingly, E. The difference between the methods having the same realizations, from the point of view of the process, maybe for example, in the way of commutation of their parameters with characteristics of the model. Not prohibited (although not recommended) just "synonyms".

Finally, we supplement the axioms of the species of structure "model-component" $R_1 - R_7$ with two more:

The axiom R_8 requires that the sets X, M, E have no identical elements. Note that the verification of compliance with this axiom is not formalized and is left to the discretion of the model developer. What should be considered the same and what differs in the model, – depends of its semantics.

The axiom R_9 sets the following rules to run any kind of object of species of structure "model-component": First, it is believed that at the beginning of the simulation step current methods-elements of all processes and all of the internal characteristics of the model are known (at the first step - there are the initial values of the internal characteristics and the initial methods-elements of processes). Secondly, observability of the external characteristics at any moment is assumed. Further,

1. The events associated with the current elements of the processes are computed. The correlation of events with the current elements of the processes is determined by the rules of switching (7). The events can be computed in parallel, but to promote the computing process further, you should wait for completion of the computation of all the events. If there are events occurred, it is checked whether there are transitions to the fast elements. If there are any - the fast elements run (they become current). They can also be computed in parallel, but to advance the computing process further, you should wait for completion of all the calculations of the fast elements, and then return to the beginning of the item 1. If there are no transitions to fast elements - the transitions to the new slow elements are made, and then return to the beginning of the item 1.
2. If there is no occurrence of the events, from all the forecasts of the events is selected the nearest $\Delta\tau$.
3. If the standard step of modeling Δt does not exceed the predicted time to the nearest event $\Delta t \le \Delta\tau$, – we compute the current slow elements with the standard step Δt. Otherwise, we compute them with the step to the nearest predicted event $\Delta\tau$. Slow elements can also be computed in parallel, with expectation of completion of the latter.
4. Return to the beginning of the item 1.

Thus, the species of structure "model-component" is fully defined and even somewhat explained.

4 Synthesis of the model-complex from models-components

Now suppose that there are two mathematical objects with species of structure "model-component". We show that the species of structure "model-component" can be extended onto the union of their base sets. First let the sets X, M, E and X', M', E', do not have the pairwise intersections, i.e.,

$$X \cap X' = \varnothing,\ M \cap M' = \varnothing,\ E \cap E' = \varnothing. \tag{9}$$

Then, obviously, the typical characterizations (2) - (8) and axioms R_1 – R_7 are valid. Indeed, the processes of the complex are the integration of the components ones, commutations - association of components commutations, switchings – components switchings association. Execution of axioms R_8 follows from (9) and from its validity for the components, and the axiom R_9, by the definition, is valid, if we perform it.

As was already mentioned above, there cannot be a formal criterion of the conditions (9). Only the developer can check their validity, basing on his understanding of the model subject area. However, if the developer has indicated that the conditions (9) are violated and concretized what this violation is, we can point to a number of formal actions to be taken for correcting the situation.

Let's start with the easiest. Let $M \cap M' \neq \varnothing$ or $E \cap E' \neq \varnothing$. For definiteness, let $M \cap M' = \tilde{M} \neq \varnothing$. Then exclude $\tilde{M}$, for definiteness from M'. Immediately begins running condition $M \cap M' = \varnothing$, but for the elements of $\tilde{M}$ will cease a part of the conditions (2) $m'_{j,real} \subset M'_j \times \tilde{M} \subset M'$ as M' does not already contain $\tilde{M}$. Then replace these mappings on the mappings in the same implementations, but already contained in M and not in M': will be $m'_{j,real} \subset M'_j \times \tilde{M} \subset M$ – that could fit into the standard form of commutation of elements with implementations of the type (2): $m_{j,real} \cup m'_{j,real} \subset (M_j \cup M'_j) \times (M \cup M')$.

Now let the characteristics intersect. Again start with the easiest. Let intersect external characteristics of our component, for example, at the characteristic $\tilde{a}$. This means that the external characteristic $a_i = \tilde{a}$ of the first model and the external characteristic $a'_j = \tilde{a}$ of the second, reflect the same (otherwise they would differ) factor of the external to our model world. From the general concepts about unity and objectivity of the external world, should follow that the values of these characteristics (of course, correctly identified) should coincide, and therefore, to the model-complex is quite enough one of them. Exclude $\tilde{a}$ for definiteness from a'. Then in all relationships switching (4) and (6), where $\tilde{a}$, should be replaced $\tilde{a} \in a'$ to $\tilde{a} \in a$, that will allow these ratios keeping the message in the model-complex. If to someone the considerations about "unity and objectivity of the world" doesn't seem too convincing, further, to resolve the intersection of the internal characteristics of the components will be offered another method which can be applied in this case as well.

Suppose now that there intersects a set of internal characteristics of one model-component with a set of external characteristics of the other. For example, suppose $x \cap a' = \tilde{y} = x_i = a'_j$. This means that if in the second model-component the characteristic a'_j was not simulated, but was a factor of the outer world, in the model-complex this

characteristic is explicitly simulated as the first model-component internal characteristic x_i. Therefore, the model-complex external characteristic of the second component $\tilde{y} = a'_j$ becomes an internal characteristic of the first model-component $\tilde{y} = x_i$. Therefore, the external characteristic a'_j should be deleted from the model-complex external characteristics, and all of the commutation (4) and (6), which include a $\tilde{y} = a'_j$, are to include $\tilde{y} = x_i$ instead.

Thus, in full compliance with the humorous programmer's proverb: "Documented bug becomes a feature", you can not only integrate the components into the complex, but also use in the simulation of the complex that some of the components, perhaps model something that without them did not simulate any others, so getting some benefits from the complexity of the model while building complexes.

Finally, suppose that there is an intersection of the internal characteristics of the model-component. Unfortunately, the concerns about unity and objectivity of the world are not convincing as in the case of the external characteristics. On the contrary, it is well known that there may be dozens of different weather forecasts or exchange rates. Let the internal characteristics of N models-components intersect in a characteristic $\tilde{x} = x_1 = ... = x_N$. Let us propose, as usual, to keep this characteristic in the set of internal characteristics of the first of component and eliminate it from the sets of internal characteristics of the other components. The task to calculate the value of this characteristic is left for an ad hoc component of the model, which has external characteristics $x_1,...,x_N$, and the internal characteristic – $\tilde{x}$. An algorithm to calculate it on - is left at the discretion of the developer. In the commutation relations (4) – (6), in those correspondences, where there are $\tilde{x} = x_i$, $i = 2,...,N$; they must be replaced with $\tilde{x} = x_1$.

Thus, the models-components allow association into a model-complex with possible (but not obligatory) commutation of some of internal characteristics of some components with certain external characteristics of others, possible addition to the initial set of components some extra to eliminate ambiguities, and correction in part of relations (4) – (6) for some components. As was shown, the resulting model-complex has a species of structure "model-component" and, thus, in turn, may be a member of new models-complexes. This way you can build models of an arbitrarily complicated fractal design, computational processes of which, however, are completely determined by the axiom R_8 of species of structure "model-component".

5 Model Synthesis vs. Object Analysis

We now consider the comparison of the fundamental concepts of model-based and object-oriented programming. First of all, note that the object-oriented approach is now presented in two forms: one that can be called "basic", it is based on concepts such as class, object, typing, inheritance, encapsulation, polymorphism[1], which are implemented some nuances in most modern imperative programming languages such as C++, Java, C#, Delphi and many others. The second, who might be called "advanced", presented by unified modeling language UML [17-18]. The UML includes all of the above basic concepts; in addition, its creators have opted for a sharp increase in the number of initial concepts and ideas. For example, apart from the "vertical" inheritance relationship that is often called a UML generalization relationship (the generalization relationship is directed to an ancestor from the child), there are relationships of association, composition, aggregation and dependence. Now it is possible to describe the behavior of systems, even in several ways: interaction diagrams, state diagrams and activity diagrams. In the UML there are many things possible to be done in a number of ways, which makes it a convenient tool for professionals, but very difficult for beginners. The authors of the language say: «UML is subject to the rule of 80/20, i.e., 80% of most problems can be solved using 20% of the UML» [17].

In contrast to the UML object analysis, the model synthesis is minimalistic in a set of basic concepts: it has the only basic concept – model-component and an auxiliary concept – model-complex, which after all, can also be a model-component. However, the model synthesis does not claim to cover the breadth of UML area. Its scope is complex multicomponent "atomistic" systems. We continue the comparison of basic concepts. Such concepts as class, object, typing in the two approaches are understood near the same. It should only be noted that the roots of these concepts certainly are not to be found in C++ nor even in Simula-67, but rather in the works of N. Bourbaki [3], the structuralists of the twentieth century [4], up to the F. Klein's Erlangen program [5]. About the same refers to the characteristics and methods.

As for the use of methods – there is a significant difference. It consists in the fact that in the object programming object and then need to call methods in the various programs, and as arguments to the method, you can pass, and take

[1] The term "polymorphism", although is generally accepted, does not seem successful, because distorts the essence of the phenomenon referred – methods overriding. Indeed, the only that stores in the object with overridden methods – is the form – ancestral characteristics and methods signatures. Thus the changes are not in the form, but in the the content of computing – what redefined methods perform. Therefore, in the future, instead of polymorphism, we will talk about methods overriding.

from him any variables that match the signature is not necessarily the characteristics of the object. In the model-oriented programming method of the model-component can work only with its characteristics, and to call it "manually" there is neither possible nor necessary – it will be invoked automatically when required by the logic of the model-component behavior. Encapsulation in the model-oriented programming is that does not allow direct access to methods. You can only operate with models-components.

This does not mean that the methods are not available. On the contrary, for example, in the realized layout of distributed simulation system [11, 12], a library of methods is published in the Internet for public use, and models-components are possible that do not have any local method, besides all of their methods can be physically located in different places of the Network. However, you can only use the method included as a part of some model-component, and inside the model-component it will not work by itself, but in accordance with the logic of behavior of its host model – such is the level of encapsulation. In some cases, it may seem more complicated, but it paid for the lack of concern about the organization of behavior of models-components – it always behaves as it can, and so its inclusion in any complex is always just a matter of proper commutation. Overriding of methods in the model-oriented programming can be easily achieved by specifying a different implementation in the ratio of typing (2), if the method is an element, or (3), in the case of an event.

Let us say a few words about the inheritance. Inheritance relationship for many classes of object-oriented programming language is a partial order. Classes that have no ancestors, but having descendants are called to them root or base. Classes that do not have descendants are called leaf. Designing with the object-oriented paradigm, of a large software system is the embodiment of the basic concepts and ideas of this system in the base classes of objects and then building a hierarchy of classes, developing, specifying and embody these ideas in a variety of leaf classes, with the help of which the target software system will be built. Such a deductive way of designing of a large software system is good when it is created "from scratch", as the world of Plato. In the simulations, however, are much more likely the problems not in the creation of new worlds, but in simulation of fragments of the existing one. In such a fragment there can easily be gathered together "The pan, the divan, the basin, the box with three locks, the valise and a tiny Pekingese"[1]. They do not appear to be deduced from the each other and to rise up to their common ancestors – is just pointless. For these tasks, "basic" version of the object approach lacks inheritance up from the bottom (in the UML such an inheritance is available). In a model-oriented programming integration of models-components to the model-complex can be considered as multiple inheritances up from the bottom.

Even if the object-oriented design using inheritance built the greatest hierarchy of classes, still all the organization of the computational process lies on the developer of the system: for the system does something - the developer is to organize calling of right methods in the desired sequence. The most difficult stage of construction of the system is not formalized – it is an art.

The attempts to formalize the process of complex software systems designing gave birth to the UML. Apparently, in fact, on the UML can be describe any system, and even from several points of view. The question is what to do next with this description – there is no unity of opinions. Some specialists (for example, [18]) believe that the main value of UML just in the application as a means of recording and sharing formalized descriptions of the stages of the sketch and design of complex systems. However, there are a number of tools allowing compiling the UML-description of the procurement classes of universal programming languages, and in this case we can speak about the mode of using UML as a programming language. However, here again we remain within the object-oriented paradigm – we obtain a hierarchy of classes and billets classes, but do not remove the need to write imperative programs to call in the correct order methods of these classes.

6 Conclusion

Organization of simulation calculations, as an invariant with respect to the integration of models-components into complexes, allows completely solve the task of complex multicomponent systems simulation models describing and synthesis. The class of models is highlighted (namely, satisfying in every point the closeness hypothesis and with piecewise-smooth, continuous on the left trajectory), for which the proposed organization of simulation calculations leads to the successful synthesis of the model. Description of a complex system model consists of a set of declarative descriptions of models-components and composed of them models-complexes. Many of these descriptions (for example, the components of one complex) do not depend on each other and so can be created and debugged by independent teams. Synthesis of simulation model of complex multicomponent system is constructed by compiling of these descriptions. Result of the compilation of any model-component (as it was shown, a model of complex system in whole – is a model-component), is a set of tables of a permanent structure in the database. In order to run the model-component, we are to put initial values of its characteristics to the database, as well as to write on universal programming languages (may be in the functional paradigm) and debug its methods.

[1] The poetry translated by Richard Pevear.

Thus, the construction of a large software system that implements simulation modeling, is divided into foreseeable stages of declarative describing of components and complexes of the model, as well as writing on universal programming languages, but in the functional paradigm, of methods-elements, and methods-events of components. Debugging declarative descriptions of the models-components and models-complexes, as well as functional programs that implement the methods, is based on the principle – debugged and forgotten. Imperative programming [14] thus is completely excluded from the project, making it easier to design and debug. The computational process of simulation (the axiom R_9 of species of structure "model-component"), is organized so that the more complicated fractal construction has the model - the higher is the parallelism degree of code produced by a program that implements the axiom R_9 – the greater is the number of methods which may be invoked in parallel. The above concept of model synthesis is applicable primarily for synthesis of complex multi-component systems. However, it is hoped that a similar approach can be used for the development of complex software systems, with the organization that fits for the paradigm described in the second paragraph, including the software systems focused on high-performance computing.

The developed concept of complex systems simulation has been realized in series of simulation models implemented under the influence of the model-Oriented programming paradigm. For example, some episodes of Reagan's SDI functioning were simulated, the model of interaction of several countries was created. Also the tools for complex systems simulation were created: the system MISS [13], and the software for the workstation of peer-to-peer system of distributed simulation [11].

7 Acknowledgments

This work was supported by the Russian Foundation for Humanities, project 12-06-00932-a, and by the Russian Foundation for the Basic Research, project 13-01-00499-a.

References

[1] N.P. Buslenko Simulation of Complex Systems *Moscow: "Nauka"*, 1978, 400 p. (in Russian)
[2] N.P. Buslenko Complex System *//An article in the Great Soviet Encyclopedia, 3-ed., Moscow: "Soviet Encyclopedia"*, 1969-1978. (in Russian)
[3] N. Bourbaki Elements of Mathematics, Theory of Set. *Springer, 2004, 414 p.*
[4] M.N. Gretsky Structuralism (philosopher) *//An article in the Great Soviet Encyclopedia, 3-ed., Moscow: "Soviet Encyclopedia"*, 1969-1978. (in Russian)
[5] F. Klein: A comparative review of recent researches in geometry *//Complete English Translation is here - http://arxiv.org/abs/0807.3161*
[6] Yu.N. Pavlovsky A Geometrical Theory of Decomposition and Some its Implementations *Moscow: CC RAS*, 2011, 93 p. (in Russian)
[7] Yu.N. Pavlovsky, T.G. Smirnova Introduction into Geometrical Theory of Decomposition. *Moscow: FAZIS, CC RAS*, 2006, 169 p. (in Russian)
[8] Yu.N. Pavlovsky, T.G. Smirnova A Problem of Decomposition in Mathematical Modeling. *Moscow: FAZIS*, 1998, 272 p. (in Russian)
[9] V.I. Elkin Reduction of Nonlinear Controlled Systems. Decomposition and Invariance under Disturbance. *Moscow: FAZIS*, 2003. 207 p. (in Russian)
[10] Yu.I. Brodsky On the model analysis, and model-oriented programming *//Vestnik MSTU Ser. Natural sciences. Spec. Issue, N3, "Mathematical modeling"* - 2013. - P. 127-138. (in Russian)
[11] Yu.I. Brodsky Distributed Simulation of Complex Systems *Moscow: CC RAS, 2010, 156 p.* (in Russian)
[12] Yu.I. Brodsky, Yu.N. Pavlovsky Developing an Instrumental System for Distributed Simulation. //*Information Technologies and Computing Systems*, N4, 2009, P. 9-21. (in Russian)
[13] Yu.I. Brodsky, V.Yu. Lebedev Instrumental Simulation System MISS. *Moscow: CC AS of the USSR*, 1991, 180 p. (in Russian)
[14] Yu.I. Brodsky, A.N. Mjagkov Declarative and Imperative Programming in Simulation of Complex Multicomponent Systems //*Vestnik MSTU Ser. Natural sciences. Spec. Issue, N4, "Mathematical modeling"* - 2012. - P. 178-187. (in Russian)
[15] I.N. Ponomarev Introduction into Mathematical Logic and Species of Structures: Textbook. *Moscow: MIPT*, 2007, 244 p. (in Russian)
[16] P.S. Laplace The System of the World. *Dublin, University Press 1830, 524 p.*
[17] G. Booch, J. Rumbaugh, I. Jacobson UML. User Guide, 2-ed., – *Addison-Wesley*, 2005, 475 p.
[18] M. Fowler UML Distilled, 3-ed., – *Addison-Wesley*, 2004, 192 p.