An Object-Oriented Architecture for Integrated CAD Systems

Vitaly Semenov*, Sergei Morozov*, Oleg Tarlapan*, Thomas Jung+

* Institute for System Programming of the Russian Academy of Sciences (ISP RAS), Bolshaya Kommunisticheskaya st., 25, Moscow, 109004, Russia
{sem, serg, oleg}@ispras.ru

+ GMD–Forschungszentrum Informationstechnik GmbH, Forschungsinstitut für Rechnerarchitektur und Softwaretechnik, Kekuléstr. 7, 12489 Berlin, Germany
tj@first.gmd.de, Tel.: 030-6392-1779. Fax: -1805

Zusammenfassung:

Eine objekt-orientierte Architektur zur Entwicklung integrierter CAD-Systeme wird beschrieben. Durch die Kombination des „Entity-Relationship"-Paradigmas mit einem objekt-orientierten Ansatz für Entwurf, Modellierung und Visualisierung ist die Architektur flexibel, erweiterbar und wiederverwendbar, so daß Komponenten mit unterschiedlichen Funktionalitäten integriert werden können und komplette CAD-Systeme für verschiedene Anwendungsgebiete auf der Basis derselben konzeptionellen, methodischen, instrumentellen und programmtechnischen Basis entwickelt werden können.

Die Architektur umfaßt einen objekt-orientierten Kernel, der invariant bezüglich unterschiedlicher Anwendungsfelder und Probleme ist, eine einheitliche Benutzerschnittstelle und erweiterte Klassenbibliotheken für spezielle Anwendungsgebiete. Der objekt-orientierte Kernel ermöglicht die Repräsentation von Ergebnissen von Entwurfs-, Modellierungs- und Visualisierungsprozessen als eine Komposition zusammenhängender Datentypen und Algorithmen. Er bietet einheitliche Mechanismen zur Komposition komplexer Objekte sowie zur Spezifikation von Modellier- und Visualisierungsszenarien und ihrer Interpretation. Mit Hilfe der graphischen Benutzerschnittstelle, die Menüs, Dialoge und Ansichten beinhaltet, können Benutzer interaktiv sowohl einzelne Objekte als auch ganze Szenarien in einheitlicher Weise manipulieren.

Im Rahmen des Aufsatzes werden mehrere Beispiele für die Benutzung der beschriebenen Architektur vorgestellt, wodurch nachgewiesen wird, daß die Architektur genutzt werden kann, um integrierte CAD-Systeme und komplexe, interaktive Applikationen für interdisziplinäre Untersuchungen zu entwickeln.

Abstract:

An object-oriented architecture for development of integrated CAD systems is proposed and discussed. Combining "entity-relationship" paradigm and an original object-oriented approach to design, modeling and visualization, the architecture offers flexibility, extensibility and reusability enough to integrate different-purpose components and to develop complete CAD systems in essentially different areas on the same conceptual, methodological, instrumental and programming basis.

The architecture includes an object-oriented kernel being invariant with respect to various applied areas and problems, unified graphic user interface and extended class libraries specific for considered areas.

The object-oriented kernel supports representation of final results of design, modeling and visualization processes as a composition of connected typed data and algorithms and provides uniform mechanisms for composing complex designed objects, specifying modeling and visualization scenarios and their interpretation. The user graphic interface includes menus, dialogs and views that permit user to interactively manipulate with separate objects and whole scenarios in uniform manner.

Discussion is illustrated by several examples of reusing the architecture for development of special systems.

The architecture seems to be promising for development of integrated CAD systems and complex interactive, graphic, computational applications intended for interdisciplinary investigations.

1 Introduction

Currently computer aided design and modeling play a central role in both science and industry technologies connected with studying complex phenomena and manufacturing high-end products. Development of complex applications, such as CAD/CAM/CAE systems, was always a difficult task connected with integrating different-purpose components; providing a wide functionality, extensibility; supporting interoperability, parallel and distributed computing; implementing convenient user interface. The integration of tools and information is a key problem to be resolved for building integrated CAD systems.

Recently product data exchange and infrastructure standards, such as STEP [1], EDA [2] developed within industry initiatives Open CAD Architecture and Interoperability (OCAI) [3], CAD Framework Initiative (CFI) [4] define interfaces that facilitate the integration of design data and tools. In combination with Common Object Request Broker Architecture (CORBA) [5] and The Reference Model for CAD-Systems (CADRM) [6], they can be applied to create distributed environments for concurrent multidisciplinary design.

The mentioned above architectures provide frameworks for the integration of different CAD applications but cannot be effectively applied for building complete multifunctional systems in view of necessity of integration of heterogeneous data and

tools within the same applications. This kind of integration has principal value for building next generation CAD systems integrating different-purpose components and providing powerful customization tools for their future evolution based on Plug&Play capability. This capability should allow extending destination and functionality of the created system directly by adding new graphic, computational, communicational, and informational components. Such integration imposes strong requirements upon uniform representation of different kinds of objects, general mechanisms for their interaction, common rules of manipulation by them.

Indeed, using an open object-oriented system, we hope that it may be relatively easy expanded by an appropriate set of specific components and, thus, may be customized for considered application areas, particular problems and technical requirements. For example, we expect that complete CAD/CAM/CAE applications integrating needed components for design specification, geometry modeling, physical processes' simulation, mathematical problem solving, visualization, rendering may be developed by such way. Nevertheless, we foresee serious obstacles to use many widespread systems for the creation of complex applications mainly because of not sufficient generality of architectures and flexibility of mechanisms by means of objects may interact with each other.

Customization toolkits such as CV DORS, PTC's Pro/DEVELOP, Matra Datavision's CAS.CADE, ACIS, Parasolids [3, 7] are stiffly oriented on particular domain models and do not permit direct expansion into other dissimilar engineering areas. For example, being oriented on the boundary representation solid model, the ACIS kernel provides expansion only in the scope of the predefined model-based core classes.

Another approach to building integrated systems has been implemented in general-purpose visualization and animation systems [8, 9]. This approach exploits object-oriented and data flow paradigms to develop reusable software components and to apply them within different visual programs. Main drawbacks of this approach are inter-connectivity mechanisms specifying rules for composition of separate objects into a scene and ways by means of which objects interact with each other while the scene is constructed. This paradigm predetermines serial composition of techniques applied to intermediate data and excludes the other possible ways of object's interaction. Nevertheless, such capabilities may be useful enough for run-time composing and interpreting complex working scenarios encountered in meaningful particular applications and, thus, may provide necessary generality for customizing an open system to specific problems. It is essential that semantics of objects and ways they interact in developed applications may not be priory known and the system architecture should provide uniform inter-connectivity and interaction mechanisms without any concretizing types of supported data and techniques.

In our research we follow to the "entity-relationship" paradigm that seems to be more natural and complete to suit generality and flexibility requirements imposed by integrating goals. Extending traditional data flow paradigm, it permits more sophisticated object's interaction schemes to be important for some topics of digital circuits design, modeling hydraulic network, simulation and visualization of bistability in semiconductor crystals considered in the paper. In combination with the

object-oriented approach it can be applied as constructive basis for general domain unified approach to development of open integrated CAD applications.

In section 2 we present an object analysis for design, modeling and visualization applications and describe the underlying principles of the unified kernel. In section 3,4,5 possibilities for reuse of the kernel for several essentially different dissimilar problems are investigated to illustrate its generality and flexibility. Section 6 is devoted to formulating a general component-based approach to unified development of complex integrated applications. In conclusions we address to more detailed information about approbation and dissemination of the proposed architecture and outdraw areas of potential applications.

2 An Unified Object-Oriented Kernel for CAD Applications

We consider the final graphic scene of a CAD application as a composition of connected typed data taking part in all processes of application, such as design specification, modeling, analysis and visualization, as well as used algorithms realizing mentioned above processes by means of constructing, transforming, deleting these data. Therefore, computer aided design can be considered as a multistep process oriented towards construction of final scene in result of defining instances of data and algorithm classes, setting relationships between them, composing working scenario from separate data and algorithms and its interpretation. To achieve the needed quality the composed scenario should be iteratively applied to the same design problem with gradually corrected data and adjusted techniques. Once being composed the scenario can be applied then to a wide range of similar problems.

We distinguish passive data-objects that control only own behavior and active algorithm-objects that can govern behavior of other objects through message passing in classic object-oriented style. An approach based on subdivision of active and passive objects reproduces to some extent Bailin's methodology known as object-oriented requirement specification [10] and has been successively applied to development of a wide range of mathematical software [16], general-purpose scientific visualization system [17], integrated modeling and visualization applications [18, 19].

In the following we consider in more details the unified object-oriented kernel for CAD applications to be developed. The object-oriented kernel (framework) is a system of abstract and concrete classes that express high-level semantics concepts of computer aided design and provide a wide functionality for building various applications on the same constructive instrumental basis.

The basic abstract class is ***Object*** that expresses arbitrary data and algorithms taking part in design specification, modeling, analysis and visualization. Objects may be saved in and restored from files, created, connected, transformed, viewed, copied, distributed over processes and deleted as application runs. To manipulate uniformly by different kinds of objects and to provide kernel functionality ***Object*** encapsulates identification key, version number, logical displacement in the scene (will be explained later) and the numbers of processes over which it has been distributed.

These attributes are shared by all scene objects, the classes which are derived from ***Object***.

Besides common attributes each concrete object ***obj* Î *Object*** has an own set of attributes defining its internal state and behavior as well as a set of typed links. Links are external ports of objects through which they may connect with other ones. Each link corresponds to some unidirectional connection of its object with other object or objects. A type of separate link ***Link* Í *Object*** predetermines potential capability of the object ***obj*** to connect with any other ones ***lobj* Î *LinkObject***, type of which satisfies to link type or, by another words, is its subtype ***LinkObject* Í *Link***.

Availability of connections in a scene means functional dependence of its objects and, consequently, necessity of their simultaneous consideration and analysis. Being set each connection defines usage relations between a main object having a link and auxiliary objects involving to it. We consider single and multiple connections. A single connection defines one usage relation between a pair of objects considered as main and auxiliary ones. Multiple connections should be used when one main object interacts with a subset of auxiliary objects of the same generic type through one link ***lobj_i* Î *LinkObject_i* Í *Link***, $i = 1,...,n$. The number of objects n involved into multiple connection may be arbitrary and depends only on particular scenario realized in an application.

To distinguish ways by which objects may interact, all links and connections corresponding to them are classified as input, output and mixed (input/output). Having links and participating in connections, a main object uses data and methods of auxiliary objects and, therefore, may change states of connected objects. It is suggested that the main object is capable to change states of auxiliary objects through output and mixed links, and is not capable to influence on input objects. The main object depends only on input objects and mixed objects and does not depend on output objects. The main object and auxiliary objects involved in mixed connections are mutually dependent. Thus, the described dependence relations based on classification of links of interacting objects may be established.

To develop parallel applications we need to refine the concept of object from an implementation viewpoint. We distinguish between objects that may be located only at any of the processes initiated by the application and objects that may be distributed over a set of the processes. Local objects are implemented by traditional methods of sequential programming. An implementation of global objects is based essentially on parallel programming. For brief we omit details connected with possible techniques for data partition and consider that each global object may be scattered and gathered in accordance with some generic rule specifying a subset of processes over which the object to be distributed and a way of its geometric, physical, logic or any else partition.

Encapsulating described behavior and properties of objects, the class ***Object*** specifies the following groups of methods common for its concrete instances:

- creation (construct, destroy, copy; save, restore in/out file),
- identification (identify an object, its class, parent class, version; verify whether the object belongs to given type),
- inter-connectivity (get number of links, their classes, types; connect objects),
- parallel processing (transmit (send, receive) a local object, distribute a global object),
- scenario information (get parameters of a scene, logical arrangement of an object in a scenario).

The basic concepts are also data-objects ***datÎ Data Ì Object*** and algorithm-objects ***alg Î Algorithm Ì Object***. Abstract classes ***Data***, ***Algorithm*** are derived from ***Object*** and inherit its behavior and properties. The class ***Data*** expresses entity of various data encountered in CAD applications. Its instances may control only the own behavior. The class ***Algorithm*** represents various algorithms, transforms, operations, and auxiliary utilities realizing all processes in computer aided design processes. The algorithms are active objects that can control both own behavior and behavior of auxiliary objects (not only data) connected through their links. The algorithms may not have input links, but necessarily have output and/or mixed links. In addition the algorithms have time attributes connected with discrete-event simulation discussed below.

The essential distinction of algorithms consists in their activity that is realized when appropriate events occur in an application. In these cases an appropriate method for running algorithm is activated. It is suggested that all connections of algorithms have been preliminary set to activation moment. In the opposite case the algorithm is considered as passive data-object and is not activated to eliminate possible error situations. While an algorithm is running, it sends messages to connected objects to get states of input and mixed objects, to perform needed operations over them, to update mixed objects and to construct output objects.

Figure 1 gives an example of an objects' interaction. The example illustrates how typed data and algorithms may be connected and interact through typed links. A hierarchy of classes specifies inheritance relations between classes of data and algorithms. A scenario diagram specifies a particular scheme of connected and interacting objects. In the scenario diagram data and algorithm instances are shown as ovals and rectangles correspondingly. Links of the objects are marked by points. Connections between objects are shown as arrows. Such scheme can be represented in terms of object-oriented methodology with usage of Booch's notation [11]. Figure 2 gives Booch's diagrams of classes and objects for the same example.

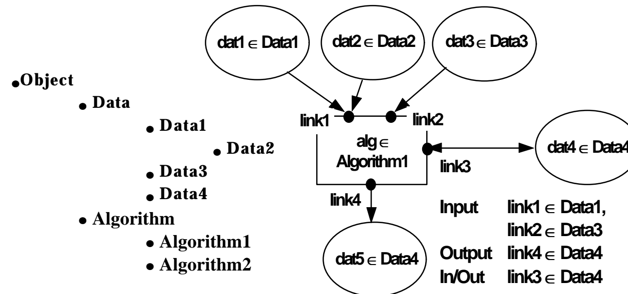


Figure 1: Class hierarchy and interaction of typed objects

Note that the accepted way to direct arrows corresponds to data flows in the scenario. This circumstance reveals similarity with traditional data flow diagram widely used in visualization and animation systems. Nevertheless, the scenario diagram expresses more general paradigm "entity—relationship" that, to our opinion, is more preferable in view of capabilities to specify more sophisticated kinds of objects' interaction. The design problems discussed below give examples of objects' interaction that could not be represented and described by data flow diagram but they are naturally specified in terms of our approach.

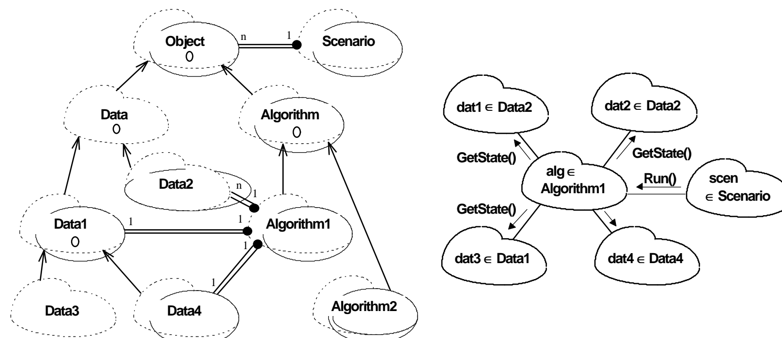


Figure 2: Class and object diagrams in Booch's notation for the considered scenario

A concept of ***Scenario*** reproduces an idea of representation of structured processes for solving complex design specification, modeling, analysis, and visualization tasks as a composition of typed data and algorithms. Class ***Scenario*** is a container class that supports ordered representation of inserted data and algorithm objects and implements functionality needed for composing and interpreting complex working scenarios without any refinement of types of particular objects.

As a rule integrated CAD systems include many different-purpose components that should be applied in combination with each other to correctly specify design requirements and features, to adequately reproduce simulated phenomenon and to realistically visualize obtained results. The described above representation allows to

connect different-purpose algorithms within the whole working scenario and then to apply it to a complex task. Since the designing process has usually iterative character, a scenario once composed may be repeatedly applied to the same task with modified data properties, adjusted techniques' parameters, more convenient views. The scenario can be refined by iterative way and be interpreted until desirable design quality will be achieved.

Instances of the ***Scenario*** class may correspond to both separate parts of composite designed object and sequences of algorithms (instructions, commands) to be performed to accomplish an appropriate design process. In the first case the scenario contains only passive data-objects, while the scenario containing active algorithm-objects can be interpreted by appropriate way for solving appropriate complex task. Indeed, as particular algorithms of the scenario are activating, its objects will interact with each other resulting in the construction of new objects, destroying and updating existing objects and generating final design object. In the following we will not distinguish these cases using the same generic class implementation.

The functionality of the class ***Scenario*** is provided by abstracting types of data and algorithms specific for particular developed applications. The class defines following groups of methods:

- creation (registry class of an object, construct an object of given class, update version of an object, copy, delete),
- identification (get an object by an identification key, select objects belonging to a given type),
- inter-connectivity (connect an object with given auxiliary objects, connect objects automatically),
- scenario interpretation (get physical and real time, iteration number; plan a scenario (rank the scene, arrange its objects), analyze latent objects, manage events; run a separate algorithm, active algorithms and whole scenario),
- processing application events.

Discuss some features in manipulating whole scenario and its interpretation.

To simplify the connecting procedure a special mechanism is provided to connect objects automatically. The mechanism is based on type analysis performed for all objects and inclusion of acceptable objects into connections marked as automatic. All acceptable objects are included in multiple connections. Only the first selected objects are included in single connections. This technique is very useful in cases when a type of an object predetermines some semantic actions that should be performed automatically. For example, it would be convenient to draw geometric objects automatically just after their construction. In this case the procedure intended for drawing the whole scene may be implemented as an algorithm having input multiple link of the geometric object type ***GeometryData*** $\subset$ ***Data***. Every time a new geometric object should be constructed, it is automatically connected to the algorithm and drawn without any additional efforts.

To perform a composed scenario different interpretation schemes can be applied. The class ***Scenario*** implements two basic schemes corresponding to logical and physical (event-based) simulation. Being used repeatedly both schemes can be applied for static and dynamic analysis of behavior of the objects to be designed, simulated and visualized. The only distinction lies in the order according to which separate algorithms of the scenario should be activated.

2.1 Logical interpretation

Following to logical interpretation algorithms are activated in the order strictly predefined by an user. Such scheme is often used in general-purpose scientific visualization systems exploiting visual programming paradigm [8, 9]. Correct interpretation of the scenario has to result in serial passing data through conveyor consisted from processing modules and to generating final results. If logical order was disturbed, there may occur errors connected with attempts to activate algorithms without preliminary prepared data. To avoid such situations the user should arrange algorithms in the right order corresponding to dependence relations arising between the scenario objects owing to links having appropriate input, output or mixed status. In sophisticated scenarios containing a lot of connected objects the arrangement may be very difficult for the user. To simplify it the class ***Scenario*** provides methods for automatic and semi-automatic arrangement of objects. The methods are based on ranking scenario objects. Ranking aims to assign to each object an index (vertical coordinate) locating its logical position in the scenario diagram similar to that presented in Fig.1. It is essentially that objects placed on diagram may depend only on states of objects located in upper positions and, conversely, do not depend on states of objects having lower positions. The orientation of arrows connecting objects through links corresponds to dependence relations between objects and to the succession of a serial interpreting scenario from top algorithms to down ones. Ranking can be easily formalized and implemented. For scenarios without cycles (feedback loops) ranking results in unique arrangement. For scenarios containing cycles the result may be ambiguous and the user should preliminary order algorithms belonging to detected cycles.

Being composed and arranged the scenario can be then executed repeatedly for new sets of data to be processed. Often significant part of the scenario is remaining without any changes. To exclude redundant running algorithms in such situations and, thus, to increase efficiency of the scenario execution a latency analysis should be performed. The latency analysis aims to select latent object — objects whose states can not be changed at current iteration. Indeed, if input data of an algorithm have not been changed at current iteration of the scenario, activation of the algorithm cannot result in changes in output objects and, therefore, has no meaning. The interpretation methods of the class ***Scenario*** accomplish the latency analysis by comparing versions of input objects with their versions stored at previous iterations. If versions of input objects have not been changed, the object and its outputs are related to latent ones and are omitted in interpreting whole scenario.

2.2 Physical interpretation

Using physical interpretation of the scenario both dependence and time relations should be taken into account to reproduce dynamic behavior of designed system, to perform time-domain analysis and to animate obtained results. Physical interpretation scheme is based on discrete events modeling and processing widely used in different applications such as digital design of logic circuits described in VHDL [12], modeling virtual reality accepted in VRML [13]. Notice that considered discrete-event simulation is applied at the level of whole scenario permitting continuous simulation at the level of separate algorithms. Within the context of this scheme, an event is considered as incident that causes the scenario to change states of its objects by means of activation of an appropriate algorithm. Each event should be processed by the appropriate algorithm in time order. A succession of events provides an effective dynamic model of the designed system being simulated.

Let us discuss implementation of the interpretation scheme provided by the ***Scenario***. To manage the events and to activate the appropriate algorithms a global event queue is supported by the class. Taking part in queue, the events are represented as pairs of algorithm identifiers and activation times. Simulated events occur only in the result of running some algorithms, generating output objects and changing mixed objects. Running algorithms create events corresponding to changes in objects' states and resulting to activation of next algorithms and generation of additional events. Thus, events are propagated along the scenario according to both dependence relations and chronological order.

We distinguish between algorithms that can generate outputs in predetermined time moments and algorithms that generate outputs only in some time interval after they have received inputs. The first type algorithms, so named generators, create output objects and corresponding events in fixed moments according to some user-assigned rules. Moment of each next creation is suggested to be known and can be obtained through appropriate method of ***Algorithm*** class. Algorithms of the second type are related to processor group. The processors generate outputs and accompanying events in some period from moment when input objects have been prepared, thus delaying data processing. Time delay can be obtained through the same method. Values returned by this method can be either deterministic or stochastic, thus, providing implementation of different kinds of simulation, including multi-factor analysis, Monte Carlo simulation, etc.

Finally, the class ***Scenario*** provides a general method for processing typical events connected with user interaction and application functionality. User events are reviewed and processed periodically combining with processing simulated events. User events are directly dispatched to appropriate methods of the scenario.

3 Example A: Logical Design and Modeling Digital Circuits

In the following we discuss possibilities for reuse of the described above kernel to design and to model digital circuits. Usually digital circuits are simulated taking into

account different design abstractions corresponding to switch-, gate-, functional- and behavioral levels [12]. In this section we will primary focus on circuits described at the gate level. At this level the functionality of circuits, such as NAND, XOR, adders and flip-flop gates, and their interconnections should be studied. Each digital element performs some elementary logic function and has fixed number of input and output terminals via which it can be coupled with other elements. It is typical for digital circuits that the connections between elements are unidirectional, which diminishes their interaction and allows applying discrete-event simulation algorithms for both DC and transient analysis.

Following to the considered above object interaction model, digital elements and signal are defined as objects. Digital elements should refer to active algorithms represented by abstract class ***DigitalElement Ì Algorithm***, and signals — to passive data represented by concrete class ***DigitalSignal Ì Data***. The class ***DigitalElement*** acts as building block for digital elements by means of specifying virtual method for calculating logic function. Methods to get numbers of inputs and outputs have been defined in superclass ***Object***. To implement particular digital elements derived classes should be developed as inherited from ***DigitalElement*** with the redefined methods implementing their particular properties. Besides common attributes the memory elements encapsulate logical variables determining their internal states. Values of the variables should take part in implementation of logical function. Possible library of digital elements is represented in Fig. 3, as a class hierarchy. Specific classes ***Generator***, ***Gauge*** should be also supplied to generate input signals for circuits and to display output results. The library can be extended in a uniform way to implement other types of used elements and signals enveloping other simulation problems for digital circuits at different abstraction levels, including mixed mode simulation. In this sense the data and algorithm class library may be considered as extended component library.

Another basic class ***DigitalSignal*** corresponds to the concept of signal in digital circuits. As an instance of this class each signal stores own digital value associated with the state of corresponding connection of a circuit. Signals have neither inputs nor outputs because elements can be coupled via own terminals having the same type ***DigitalSignal***. Signals are passive objects whose states can be changed by activating appropriate digital elements. Being activated elements get states of input signals, perform own logical functions and put calculated values into output signals.

It is obviously that any digital circuit can be represented as composition of the enumerated basic types of objects (or scenario) that unambiguously specifies digital elements of the circuit and its topology. Discrete-event simulation of the circuit can be accomplished by means of physical interpretation of the scenario provided by the kernel.

- *Object*
    - *Data*
        - *DigitalSignal*
    - *Algorithm*
        - *DigitalElement*
        - *Not*
        - *And-Not*
        - *Or-Not*
        - *And-Or-Not*
        - *Multiplexer*
        - *Adder*
        - *RAM*
        - *ROM*
        - *Delay*
        - *Generator*
        - *Gauge*

Figure 3: Component library for logical modeling digital circuits

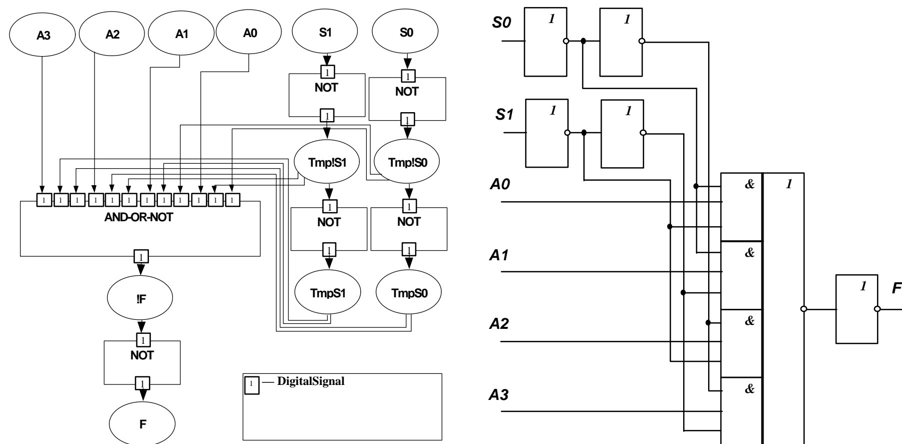


Figure 4: Multiplexer and its scenario diagram

Figure 4 gives an example of a digital circuit intended for multiplexing four signals. The circuit has been designed on logical elements ***And-Or-Not***. Input control signal "S0-S1" and data signals "A0-A3", intermediate signals as well as output signal "F" are represented in the scenario by appropriate objects belonging to the class ***DigitalSignal***. Digital elements of the circuits are represented by objects of appropriate algorithmic classes ***Not*** and ***And-Or-Not***. All elements have terminals of the type ***DigitalSignal*** so that they can refer to inputs, outputs and connect with each other.

Thus, mechanisms of interconnection and interaction of objects supported by the kernel allow to specify digital circuits as composition of elements and signals and to apply physical interpretation for their discrete-event simulation.

4 Example B: Modeling Flows and Design of Hydraulic Networks

The class of hydraulic networks' modeling tasks is very wide and includes simulation of different engineering systems providing distributed consumers with heat energy, water, fuel or any other transported liquid or gas substances [14]. Examples of hydraulic networks are turnpike gas and oil pipelines, canal and irrigation systems, city gas, water, heat supply systems, heating and ventilating systems in buildings, etc.

- *Object*
  - *Data*
    - *HydraulicCircuit*
    - *HydraulicNode*
    - *HydraulicElement*
      - *Pipe*
      - *Tributary*
      - *Consumer*
      - *Pressure*
      - *Pump*
      - *Flow*
      - *Reservoir*
      - *Damper*
      - *Regulator*
        - *PressureRegulator*
        - *FlowRegulator*
      - *Manometer*
      - *FlowMeter*
    - *MathematicalData*
      - *Vector*
      - *Matrix*
      - *Function*
  - *Algorithm*
    - *PipingModelingAlgorithm*
      - *StaticPipingModelingAlgorithm*
      - *DynamicPipingModelingAlgorithm*
    - *MathematicalAlgorithm*
      - *LinearSystemSolver*
      - *NewtonSolver*
      - *ODESystemSolver*
    - *DrawHydraulicCircuit*

Figure 5: Component library for modeling hydraulic networks

The hydraulic system (circuit) is a composition of special devices and pipes or canals connecting them and realizing transporting compressed and uncompressed liquids (water, oil, gas, air, etc.). In any hydraulic system there are three groups of components: pressure and substance sources (pumps, compressors, accumulating capacities) providing flow of transported substance and introducing energy in the

system, pipelines delivering this substance and consumers. The source data for mathematical modeling a hydraulic system are its technical parameters, such as diameters and lengths of pipes, section sizes of canals, roughness, as well as boundary conditions, such as varied input values of flows and loads. The state of a hydraulic system is defined by the substance flows (expenses) in its components as well as pressure and temperature in connecting nodes. In contrast to the previous problem of modeling digital circuits the states of the hydraulic components are tightly connected and should be taken into consideration simultaneously. Flow of liquid or gas in a hydraulic network is described mathematically by a nonlinear system of algebraic or differential equations which represent functional characteristics of separate network components in the form of nonlinear algebraic or differential relations and fundamental conservation laws for mass and energy of transported substance in the form of linear relations similar to Kirchhoff rules for electrical circuits.

The library for modeling of hydraulic networks may include classes implementing hydraulic circuit properly, its nodes, different hydraulic elements and devices, algorithms for static and dynamic analysis of hydraulic networks, solvers of mathematical problems, mathematical data used in them for representation of intermediate data and results, algorithms for visualization of circuits and modeling results in the form of inscriptions, graphs, tables. The possible hierarchy of the library classes is presented in Figure 5.

The following example illustrates how a complex scenario intended for dynamic analysis of two-thread oil pipeline (Figure 6) can be composed and interpreted within a hydraulic networks' simulation application based on the described above kernel extended by the appropriate applied class library in accordance with general methodology.

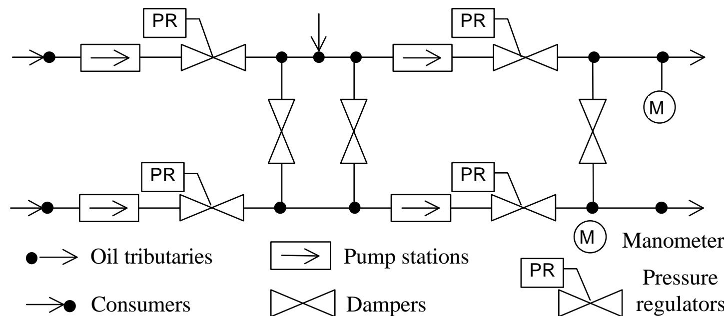


Figure 6: An example of two-thread oil pipeline hydraulic circuit

The scenario diagram is presented in Figure 7. Note that the pipeline upper thread only is shown in the diagram to simplify the figure. The hydraulic circuit has been composed within special container object of the class *HydraulicCircuit* to represent and to manipulate the circuit as separate unit. The simulation of the circuit will be performed by applying composition of algorithms for solving standard mathematical problems. Being composed in separate branches the algorithms accomplish serial reduction of the initial differential problem to non-linear and linear algebraic

problems. Since the input links of the algorithms are not defined they interpreted as data objects taking part in internal processes activated by the modeling algorithm. The object of the class *DynamicPipingModelingAlgorithm* is used for forming the mathematical model of the circuit as a system of ordinary differential equations. The modeling state values for selected components are generated and visualized through the special *DrawHydraulicCircuit* algorithm.

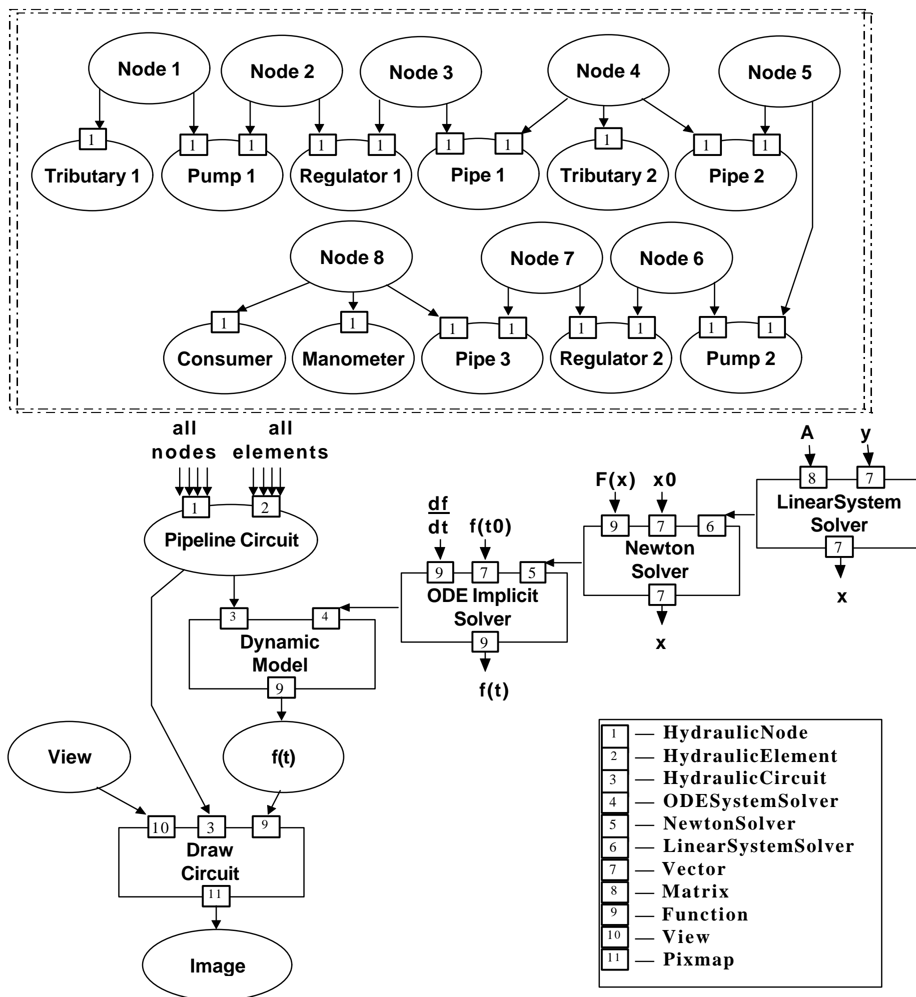


Figure 7: An example of the scenario for dynamic analysis of the oil pipeline

5 Example C: Simulation and Visualization of Semiconductor Crystals with Optical Bistability

The considered kernel can also be reused for specifying and interpreting different scenarios for simulation and visualization of various computational mechanics problems. To this end it should be extended by an applied library implementing various scientific concepts as well as widespread methods and techniques intended for visualization of these problems. The library can be organized as an integral hierarchy of concrete and abstract classes as presented by Figure 8. The library includes classes for manipulation with different kinds of data such as structured and unstructured surface and volume meshes, polylines, point sets, physical fields, color palettes, scales, glyph sets, orthogonal slices, views. The library includes classes for interpolating fields, extracting isolines and isosurfaces, constructing streamlines and streamtubes, tracing particle trajectories, orthogonal slicing, constructing glyphs, calculating field norms.

Figure 8: Component library for scientific visualization

This component repertoire corresponds to functionality of general-purpose visualization systems such as AVS, Data Explorer, and IRIS Explorer [8]. The used abstractions permit to manipulate uniformly different geometry and topology data types and to draw them in an OpenGL context. Drawing rules specify how arbitrary geometry data should be drawn by separate topology elements and colored in accordance with assigned palette. The physical field class may support generic multi-component representation corresponding to scalar, vector, and tensor fields given at different kinds of multidimensional geometry data.

In the following we consider in detail an example of a scenario intended for visualization of results of modeling switching processes in optical bistable semiconductor crystals. This problem is important for the construction of new semiconductor devices being a base for future optical computer and network systems. Its strict mathematical statement and details of simulation technique can be found in [15].

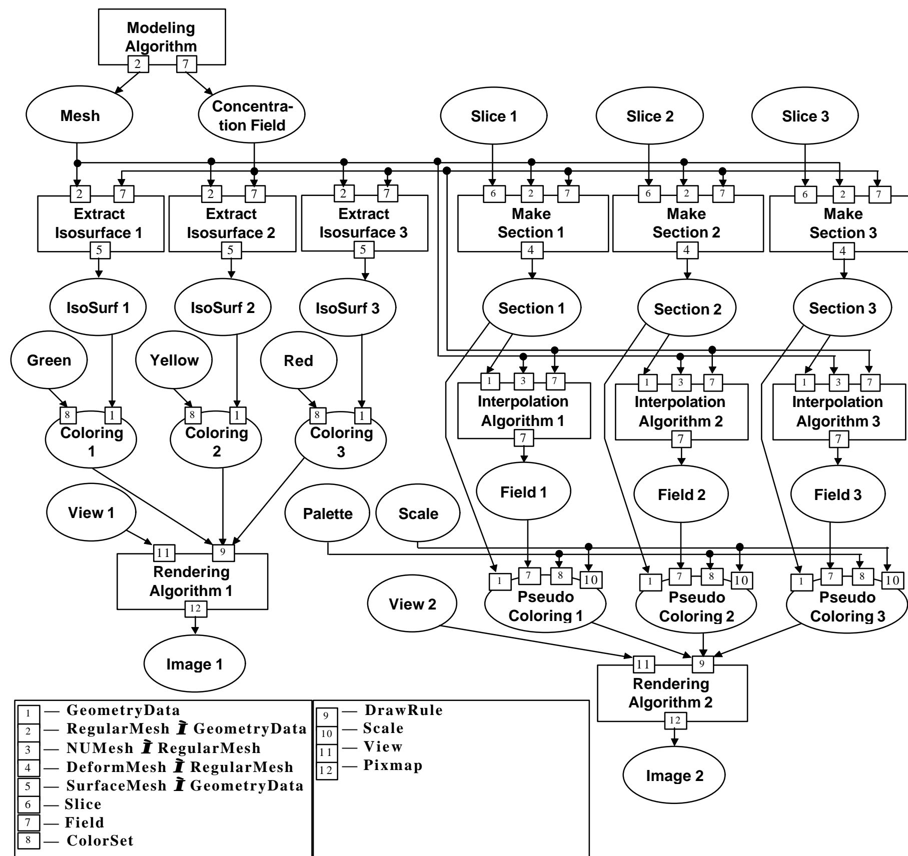
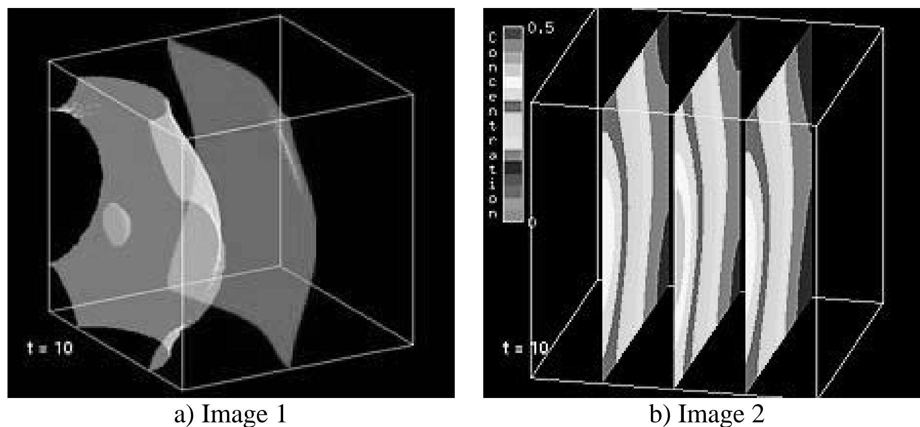


Figure 9: Diagram of the scenario intended for studying the switching processes in optical bistable semiconductor crystals

In the presented example the process of charge carrier distribution transition to steady state in optical crystals is investigated in dependence on laser radiation and carrier diffusion. An optical bistable element has two steady states. The first state (logic "0") is characterized by full light transmission through a crystal and small charge carrier concentration. The second state (logic "1") is characterized by decreasing greatly output light intensity and increasing charge carrier concentration. The following information is very important for developers of optical systems (in particular, memory devices) based on such elements: guaranteed "up"-switching (it is problematical when charge carrier mobility is high), time of this switching, profile of information light beam intensity at the output from the crystal (symmetric profile improves quality of switching registration and allows to decrease sizes of the element), spatial localization of the second steady state.

The algorithm of the class ***ModelingAlgorithm*** performs the simulation process and generates output data as regular mesh and charge carrier concentration field assigned to it. To investigate the dynamic process of "up"-switching (transition of the crystal to the second steady state), to determine switching time and spatial localization of the second steady state it is effective to apply isosurface extraction technique to the discussed problem. Three isosurfaces corresponding to low, medium and high level of charge carrier concentration are extracted and colored by different colors of a rainbow palette. It is also convenient to apply pseudo-coloring technique on orthogonal slice sections of the crystal cube with planes parallel to the laser beam direction. To pseudo-color the obtained orthogonal sections the source concentration field is interpolated on them. Drawing rules are used as the input parameters of the rendering algorithm that performs joint analysis and drawing geometry objects of the scene in accordance with specified rules in the given view. The order of applying the algorithms and obtaining the intermediate results is easily observed over the scenario diagram presented in Figure 9. The obtained final images are given by Figure 10.



a) Image 1 b) Image 2

Figure 10: Final images for the problem of optical bistability in semiconductor crystals

Note that most of the visualization techniques allow generic implementation, which does not require strict concretizing input data types. For example, in the presented diagram the isosurface extraction algorithm is implemented using regular mesh data abstraction. In that case the algorithm can be applied in the same manner both to uniform, non-uniform orthogonal meshes and to deform meshes. The attraction of the abstract data type mechanism simplify significantly composing complex scenarios using relatively small repertory of techniques and is significant for functional extension of the applied library, because addition of new data types does not lead to necessity of development of new algorithm classes.

6 Component-Based Approach to Building Integrated CAD Systems

The considered object-oriented kernel is general and flexible enough to be used for development of various CAD systems on the same conceptual, methodological, instrumental and programming basis. Combining the "entity-relationship" paradigm and an object-oriented approach the kernel reproduces a general object model suitable for many applied problems, for example, assembling CSG models in machinery CAGD systems, construction of visualization pipelines in general-purpose scientific visualization systems, specification of combined computational strategies in mathematical systems, composing geometric scenes in virtual reality modeling browsers and animation systems, modeling network flows in CAD systems for piping systems, electron devices. Being based on the same model various working scenarios can be composed from separate instances of data and algorithm classes and can then be applied for solving mixed design specification, modeling and visualization problems.

Indeed, to apply the kernel one should not modify it any time to suit to a particular problem, semantics of specific entities, their relationships, and possible details of program implementation. Supported mechanisms of interconnection and interaction of objects allow composing and interpreting sophisticated scenarios encountered in real applications.

These circumstances allow us to consider the kernel as a framework for building complex integrated applications and to formulate general component-based approach to unified development of CAD systems and their families in essentially different applied areas. Principal demands imposed upon design of such systems are the following:

- the system should have an open architecture allowing functional extensions, various working scenarios, parallel and distributed usage;
- the destination of the system should be completely determined by semantics of registered components and their collection;
- the functionality of the system should be provided by capabilities for run-time composing working scenarios and their concurrent interpretation.

The proposed component-based approach consists of reuse of the open software architecture and of development of applications including the object-oriented kernel, an unified graphics user interface and expanded problem-oriented class libraries. C++ language, the graphic library OpenGL and standard GUI libraries (such as Motif, Qt or Gtk) may be used as standard implementation tools permitting to port developed applications to different platforms with minimal efforts. The common open architecture of an application is shown in Figure 11.

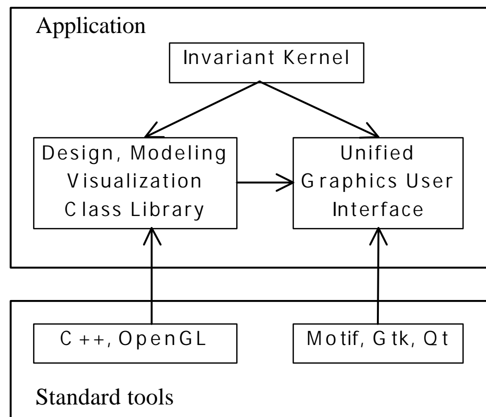


Figure 11: Architecture of an application

To provide invariance of the kernel its implementation should be based on meta-object information specifying in generic form properties and behavior of particular objects that may be encountered in applications. The same information can be effectively utilized for the construction of an unified graphics user interface including different kinds of menus, dialogs, toolbars, toolboxes, status bars, 3D views. The elements should permit to represent object repertoires, to compose and to interpret scenarios in the scope of object-oriented and visual programming paradigms, to save and to restore generated scenes and composed scenarios to/from files. For example, completely unified elements may display the hierarchy of registered classes, the current scenario diagram, the filtered list of scenario objects. Note that manipulation with particular objects can be carried out also through an unified dialog that supplies edition elements for public attributes, methods and references listed in accordance with meta-object information for the object class. Through the dialog the user can set desirable values of attributes, run public methods and connect objects via links. In this sense the unified interface is adaptable to repertoire of components registered in the application. For graphic applications the interface may include additionally 3D views for rendering geometry scenes and toolbars for manipulating separate views and geometry objects.

Following to the proposed component-based approach building an application should be reduced to object analysis of interesting application area, to selection of specific data and design, modeling, visualization techniques entities, to the development and

implementation of an component library and to its registration in application template. This process can be accomplished following the iterative scheme:

- identify classes of scientific data and algorithms taking part in statement and solving design, modeling and visualization problems by selecting key abstractions of the applied area;
- define semantics of selected classes, reveal generality relations between them, such as “general—particular”, “integral—part of”, develop object classifications (hierarchies of inherited classes) for data and algorithms with tops in the classes ***Data*** and ***Algorithm*** correspondingly;
- establish usage relations between instances of classes by analyzing semantic dependencies and represent them as input, output and mixed links of objects, subdivide single and multiple links;
- identify specific attributes, properties and behavior of particular objects, implement their classes, for concrete algorithm classes implement the specific running method;
- compose and test different scenarios for the considered application area with chosen objects, reconsider the object system critically, expand functionality of existing classes, try to select new entities, and, possibly, return to preceded development steps.

Taking into account the benefits from object orientation, the implementation of the library can be significantly simplified and improved through exploiting principles of inheritance and polymorphism in development of component classes.

It is important that the described unification of the object-oriented kernel and the interface of applications does not hinder their specialization for particular purposes. For example, in equal degree they can be used for both development of simple applications with built-in scenarios oriented on end-users and building complex parallel applications with run-time altered scenarios intended for high-performance computing. Some details connected with building parallel and distributed applications in the scope of the same approach can be found in [17].

7 Conclusion

Thus, the open object-oriented architecture for building complex integrated applications has been proposed. Generality and flexibility provided by the architecture and the formulated component-based approach allow to build complex integrated CAD systems and their families for essentially different dissimilar scientific and industrial areas on the same conceptual, methodological, instrumental and programming basis.

The class of potential applications is extremely wide and envelops numerous CAD/CAM/CAE systems, GIS systems, mathematical systems, modeling applications

for multidisciplinary research, scientific visualization systems, virtual reality modeling browsers, and animation systems.

Early, the architecture and the approach have been successively approbed in development of systems for constructive solid geometry modeling, scientific visualization, computational mechanics within OpenModeler&Visualizer (OpenMV) programming environment [16-19].

Besides domain model unification, an important advantage is that the architecture allows building concurrent CAD systems in parallel and distributed environments for solution of large-scale engineering problems and collaboration between partners involved in joint design projects. This capability is achieved by more wide interpretation of basic concepts and by implementation of local and distributed data and algorithms, scheduling methods, and communication facilities as specific components. By such way both OpenMV-based and non-OpenMV-based systems can be integrated.

The discussion of aspects concerning building parallel systems within OpenModeler&Visualizer can be found in the paper [17] that presents parallel implementation of the volume visualization system for high-performance computing.

Planned works will be directed on building collaborative design applications in interdisciplinary areas.

8 References

1. External Representation of Product Definition Data (STEP), ISO DP 103030, 1989.
2. Standards for Electronic Design Automation, released by CFI, Version 1.0.0-112592, 1990.
3. Jeff Simon , Open CAx, STEP & Objects, http://www.stellar.org/global/e-eng3.html
4. Guy E.T. An Introduction to the CAD Framework Initiative, Electro 1992 Conference Record, Boston, Massachusetts, May 1992.
5. The Object Management Architecture Guide. Object Management Group, 1990, Revised 1995.
6. Abeln O. CAD-Referenzmodell zur arbeitsgerechten Gestaltung zukunftiger computergestutzter Konstructionsarbeit, 1.Ed., Stuttgart, 1995.
7. ACIS Technical Overview, ACIS Geometric Modeler, Programming Manual, ACIS 3D Toolkits, Spatial Technology Inc., Boulder, CO, 1992.
8. Jern M. Information Visualization — Trends in the late 90s // Proceedings GraphiCon'96, Vol. 1, pp. 91–131, Saint-Petersburg, 1996.
9. Object-Oriented and Mixed Programming Paradigms: new directions in computer graphics / P. Wisskirchen (ed.), Springer, 1996.
10. Bailin S.C. An Object-Oriented Requirements Specification Method, Comm.ACM, Vol. 32, No. 5, 1989, pp. 608–623.
11. G. Booch, Object Oriented Design With Applications, Behjamin/Cummings Publishing Company, 1991.
12. Chang K.C. Digital Design and Modeling with VHDL and Synthesis, IEEE CS Press, Los Alamitos, CA, 1999.
13. The Virtual Reality Modeling Language. ISO/IEC 14772-1:1997.
14. Merenkov A.P., Khasilev V.Ya. The theory of hydraulic circuits, Moscow, Science pub., 1985 (in russian).
15. Yu. N. Karamzin, T.A. Kudryashova, S.V. Polyakov and I.G. Zakharova. Simulation of 3D optical bistability problem on parallel computer systems. Matematicheskoe modelirovanie, in «Fundamental physical and mathematical problems and modelling of technical and technological systems» (ed. L.A.Uvarova), pp. 117-124. Published by Moscow state technology university «Stankin», Moscow, 1999.
16. Semenov V.A. Object systematization and paradigms of computational mathematics // Programming and Computer Software, No. 4, 1997.
17. Ivannikov V., Morozov S., Semenov V., Tarlapan O., Rasche R., Jung T. Parallel object-oriented modeling and visualization in OpenMV environment // Proceedings of GraphiCon'99, Moscow, August 26 - September 1, 1999, pp.206-213.
18. Semenov V.A., Morozov S.V., Tarlapan O.A. The system OpenModeler&Visualizer and its applications for computational mechanics, Collected abstracts of X Anniversary International Conference on Computational Mechanics and Modern Applied Software Systems, Pereslavl-Zalessky, June 7-13, 1999, pp.308-309.
19. Klimenko S., Nikitin I., Burkin V., Semenov V., Tarlapan O., Hagen H. Visualization in string theory // ACM Computer Graphics, 1999 (to appear).
20. The ISP RAS Scientific Visualization Group Home Page, http://www.ispras.ru/~3D