STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

A COMPONENT BASED APPROACH FOR SCIENTIFIC VISUALIZATION OF EXPERIMENTAL DATA

DUMITRU RADOIU AND ADRIAN ROMAN

Abstract. The paper addresses the issue of component-based scientific visualization systems as a solution to most of the standard/commercial visualization system problems. The visualization process is requested to meet the process validation criteria. The visualization system is requested to observe a reference model. The benefits are discussed on a detailed component based visualization system.

Keywords: scientific visualization, software components, visualization reference model, and visualization process validation

1. Introduction

The commercial visualization systems, the so-called "turn key systems, are easy to use but they have some disadvantages [1]:

- they are expensive;
- they run on pretentious platforms;
- they are "rigid, meaning that one can not use more than one instance of a visualization module. Several instances of the same module allow the simultaneous execution of several sets of input data;
- their modules can not be used to build other systems;
- the implemented algorithms are not always the algorithms desired by the user;
- the functionalities of a turn key system are "fixed, they can not be changed by the user.

This article proposes a new approach of the visualization field. Three goals are to be reached:

a.: Description of a **reference model** for the visualization process;
b.: Formulation of some criteria for the **validation of the scientific visualization process** ;
c.: Description of a **software component** model used to build personalized visualization systems. Such a model allows the rapid construction of visualization systems using modules that implement the desired algorithms.

The **reference model** represents an abstract view of the visualization process. It uses the concept of **level**. A level is seen as a distinct stage of the process that accepts data and services of a certain format as input. The result is a new set of data and services offered as input to the next level.

The reference model also serves to standardize the terminology, as well as to compare the visualization systems and to identify the constraints imposed by the process.

The **validation of the scientific visualization process** determines whether the process results in a *scientific* visualization.

The model of the **component-based visualization systems** is supposed to suggest some formalization capable to offer proper answers to the following questions: How does a software component behave? How are the interfaces between two components described? Which are the conditions that allow the composition of two or more components?

The idea of the component-based visualization systems is to use independent components to construct personalized visualization systems in order to obtain optimum and flexible systems that include a large variety of functionalities. A very important advantage introduced by this model is that the visualization systems can include user-made components, implementing the desired algorithms. The construction of components implies a large agreement upon the different data models, upon a formal model for time-sometimes the time being a critical variable–, upon a user model, etc.

2. Towards a reference model

In order to describe the reference model for the visualization systems a "decomposition" of the visualization process is performed. The identification of the "levels" used for the data processing allows a better understanding of the conditions that must be imposed on the visualization function. The proposed model [2] contains three levels: **modeling level, logical visualization level** and **physical visualization level.**

The **modeling level** only "extracts" from the data set the information of interest, allowing a global processing of the data. This level performs operations such as extraction of the data geometry, sub-sampling or re-sampling, extraction of the data set characteristics, etc.

The data objects are passed to the next level, the **logical visualization level,** where they are "factorized" into primitive data and mapped to glyphs and graphic primitives. A set of functionalities is implemented at this level, e.g. the choice of the visualization primitives, of the optical characteristics of objects, the computation of the optical properties of scenes, the light settings etc. Interaction elements are also present, performing rotations, translations, zooming.

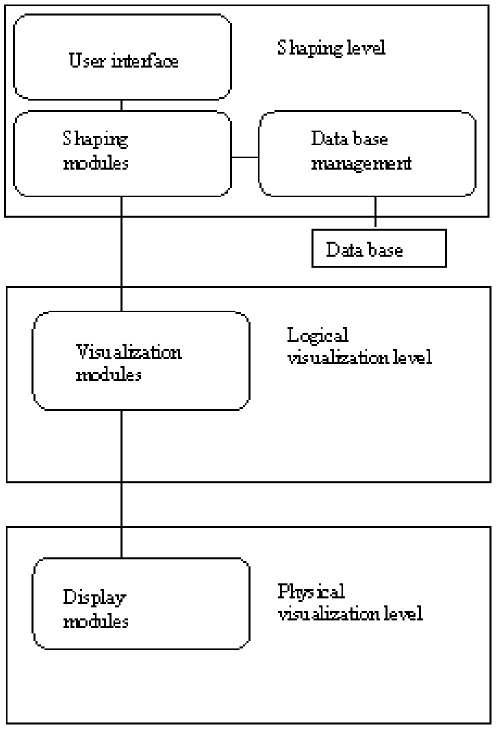

Fig. 1 The architecture of a visualization system for experimental data.

The **physical visualization level** includes the choice of visualization medium and classical tools are available to perform the hidden surface removal, shading, lightning, etc.

Figure 1 proposes the architecture of a visualization system for experimental data following the above-described model.

3. Validation of the scientific visualization process

Scientific visualization is a computational process that transforms scientific data in visual objects [3]. Not all visualizations are scientific ones. A scientific visualization guaranties a certain degree of accuracy. In order to state conditions to be fulfilled by a scientific visualization, some mathematical structures over data sets are introduced.

3.1. **Basic concepts and definitions.** The idea of using the mathematical structures defined over data sets to find conditions imposed on the visualization function has been promoted by many authors [4], [5], [6], [7], [8].

Scientific data can be obtained in many different ways, e.g. by running a simulation or through a DAQ process. Usually, scientific data objects are finite representations of complex mathematical objects. We note by $\mathbf{O}$ the set of such objects, $o \in \mathbf{O}$. During the visualization process, initial data objects, o, are processed through different transformation functions $Mat(o) = o'$, into a new set $o' \in \mathbf{O}'$. Objects o' are then mapped $Map(o') = g$ into a set of ideal geometrical objects $g \in \mathbf{G}$, through a set of graphical primitives. Objects g are usually n-dimensional (nD), animated (t) and interactive. A group of g objects is usually called the **logical visualization** of a scene.

Ideal geometrical objects g, nD, animated (t) and interactive are usually represented $Rep(g) = g', g' \in \mathbf{G}'$, on real $2D$ screens. A group of g' objects is usually called a **physical visualization** of a scene. Functions $Rep(g) = g'$ implement classical graphical operations such as composition of the scene, volume generation, isosurface generation, simulation of transparency, reflectivity and lighting conditions, $nD \rightarrow 2D$ projection, clipping, hidden surface removal, shading, animation (t), setting user interactivity (zoom, rotate, translate, pan, etc), etc.

By **interactivity** we understand the attributes of visual objects (logical and/or physical) whose setting allows $nD \rightarrow 2D$ projection (zoom, rotate, translate, pan, etc), animation control (t), control of the objects composing the scene and control of the scene as a composite object.

The scientific visualization process is described by the $Vis(o) = g'$, $Vis(o) = Rep(Map(Mat(o))) = g'$ function.

The above concepts and notations are synthesized in the table 1 (see Appendix).

3.2. **Fundamental conditions of scientific visualization.** There are many requirements concerning a certain process of scientific visualization. Here are the three fundamental ones.

The first one is the **distinctiveness condition.** This condition (although very weak) enables users to distinguish between different data objects based on their display. The condition is necessary as one can imagine many visualization functions that generate images with no use, which reveal none of the data objects characteristics/attributes.

The second condition is the so called the **expressiveness condition.** It assures that the attributes of the visual object represent the attributes of the input data set.

The third one is the **precision condition.** This condition insures that the order among data objects is preserved among visual objects.

The distinctiveness condition. em Different input data (different mathematical objects) have to be mapped into different visual objects.

This can be stated as: $o_1 \neq o_2 \Leftrightarrow Vis(o_1) \neq Vis(o_2) \Leftrightarrow Rep(Map(Mat(o_1))) \neq Rep(Map(Mat(o_2))) \Leftrightarrow g'_1 \neq g'_2$ for any $o_1, o_2 \in O, g'_1, g'_2 \in G'$

The interpretation of this condition is that $Vis()$, $Mat()$, $Map()$ and $Rep()$ functions are injective.

The expressiveness condition. *The visual objects express all and only the characteristics of input data.*

It results that the visualization function should be bijective/one to one.

The two conditions are necessary but not sufficient. Another condition is needed to establish an order relation, seen as a precision relation.

The precision condition. *For any objects o_n, $o_m \in O$ such that o_n is "more precise" than o_m we have $Vis(o_n)$ "more precise" than $Vis(o_m)$,with $Vis(o_n)$, $Vis(o_m) \in G'$.*

The precision relation adds something new. If the visualization function is well defined and the input data objects are strictly ordered, the visual objects can be ordered by precision.

The first two conditions introduce criteria of validation and control of the visualization process. The visualization function $Vis()$ fulfilling these criteria results in a scientific visualization. The third condition allows further developments by defining mathematical operations on the given ordering.

The mathematical structures and the other notions presented above are later used to support the main ideas of this paper.

4. Component based visualization systems

The alternative to the acquisition of commercial visualization systems and their parameterization to fit to the user's problem is the use of components. These are pre-made complex functional entities, that can be (re)used to construct the desired architecture of a visualization system. A component performs a specific visualization task, which can be isolated inside the system.

A component can be a class or a collection of classes. Unlike classes, a component can be implemented using a technology that is completely different from the object oriented one, e.g. an assembling language. Classes are some how similar to components, but the object-oriented technology has not imposed or hasn't been able to impose components. One of the reasons is that the definition of the objects is a purely technical one. An object represents the encapsulation of a state, of a behavior, polymorphism and inheritance. The definition does not include the notions of independence and forward composition. Components as well as objects are re-usable. Components' description has to be more general in order to be capable to assure their independence and reusability. Finally, the interaction between components has to lead to a functional system.

Considering the reference model introduced in the section 2, we conclude that a component-based visualization system resembles figure 2, where each rectangle represents a component.

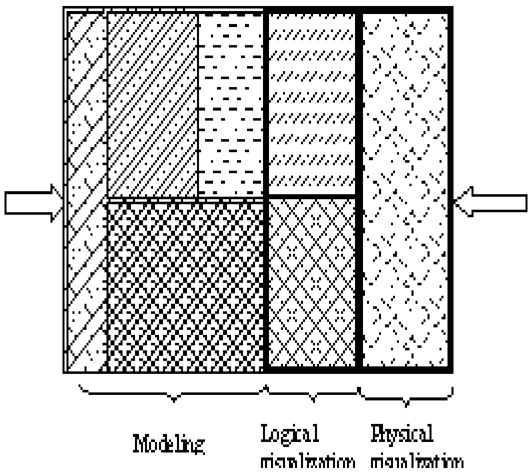

Fig. 2 The architecture of a component based visualization system

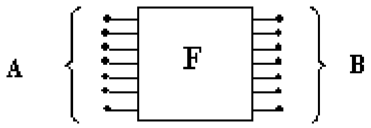

Fig. 3 The graphical description of a software component

4.1. **Software Components.**

4.1.1. *Basic Notions and Definitions.* A *software component* is a binary program that represents the physical encapsulation of related services according to a published specification. [9]

A software component can be seen as an interactive system that communicates asynchronously through channels. The services implemented can be accessed through a consistent and published interface that includes an interaction standard. A component has a *black-box* view captured by the published specification, and a *white-box* view showing implementation details.

The interface defines **a set of channels C**, divided in **a subset** $A = A_1, A_2, ..., A_n$ **of the input channels** and **a subset** $B = B_1, B_2, ..., B_m$ **of the output channels,** meaning that $\mathbf{C} = \mathbf{A} \bigcup \mathbf{B}$. A component can be described graphically as in figure 3.

Any component implements a function F :

$$F : X^n \to Y^m, \;\; (x^1, x^2, ..., x^n) \to (y^1, y^2, ..., y^m)$$

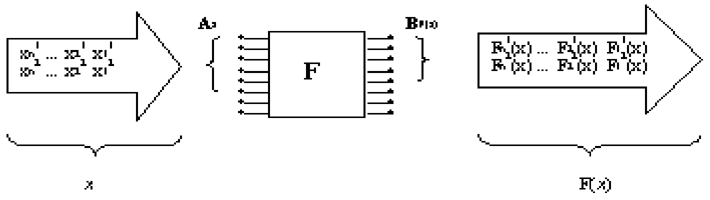

Fig. 4 The way a software component works

which assigns an output on m channels to an input on n channels. In other words it assigns the output data $(y^1, y^2, ..., y^m)$ to an input data $(x^1, x^2, ..., x^n)$. The inputs are timed streams of messages. The component sends new outputs every time the inputs change. This is why a software component can be seen as a system that communicates asynchronously (the component waits for a message; when it is received the component processes it and the result is sent to the output; then, the component waits for the next message). A message can be a string of characters, a binary number, a decimal number etc. Each channel has a stream of messages attached, representing all the messages received (sent) through that channel.

A component needs a time t in order to process a message. We consider that the inputs are introduced at the given moments: $t_0 = 0$, $t_1 = \tau$, $t_2 = 2\tau, ..., t_n = n\tau$. That means that the n-th message is accepted by the component only at the t_n moment.

Consider the input stream of messages x. The following notations are introduced:

- $x(n)$ - the string of the first n messages (until the moment t_n) from x;
- x_n^i - the input message on the channel$i$ at the moment t_n;
- $F(x)$ - the *stream* of messages assigned to the output channels for the input x;
- $F(x)(n)$ - the first n messages from $F(x)$;
- $F_n^i(x)$ - the output message on the channel i at the moment t_n;
- F_x- the *set* of messages assigned to the output channels for the input x;
- A_x - the subset of the input channels for which the input is different from zero, assigned to the input $x(A_x \subseteq A)$;
- $B_{F(x)}$- the subset of the output channels for which the output is different from zero, assigned to the output $F(x)(B_{F(x)} \subseteq B)$.

The way a component works is described in Fig. 4, or simplified in Fig. 5.

The following definitions are introduced:

(a): *The function F is* **consistent** *if for$x = z \Rightarrow F(x) = F(z)$, for any streams of messages x and z.*

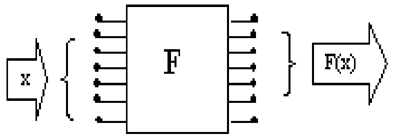

Fig. 5 A simplified scheme of the way a software component works

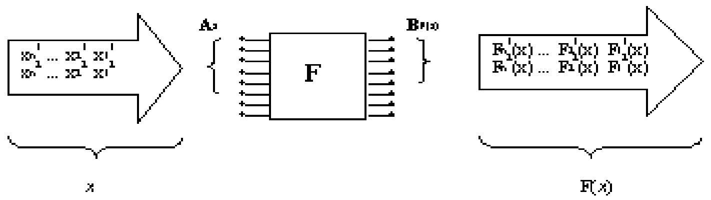

Fig. 6 A simplified scheme for a deterministic behavior of F

The function F is consistent if for identical inputs, identical outputs are obtained.

(b): *The function F is* **causal** *if for any n natural the following condition is fulfilled: $x(n) = z(n) \Rightarrow F(x)(n) = F(z)(n)$.*[9]

If the function F is causal then it is also consistent. The processing time for a causal function is $\tau = 0$. Such a component can not be constructed. We call it *ideal*.

(c): *The function F is* **strictly causal** *if for any n natural the following condition is fulfilled: $x(n) = z(n) \Rightarrow F(x)(n+1) = F(z)(n+1)$.*[9]

We consider $F(x)(0) = F(z)(0)$.

A real component is always strictly causal, meaning that the output at the moment t_{n+1} corresponds to the input at the moment t_n. This component can be implemented because the processing time is $\tau > 0$.

(d): *The function F is* **realizable** *if there is a function strictly $f : X^n \to Y^m$ such that for any input x we have$f_x \in F_x$ (the output determined by f belongs to the set of the outputs determined by F).[9]*

We denoted by f_x the set of messages assigned by the function f to the output channels, for the input x.

A function f with $f_x \in F_x$ for any input leads to a deterministic behavior of the function F (Figure 6).

(e): The function F is **fully realizable** if it is realizable and for any input x there are **p** functions strictly causal $f_x^i : X^n \to Y^m$, $i = 1, ..., p$ such that [9] $F_x = \cup_{i=1}^{p} f_x^i$

This property guarantees that for every output there is a strategy (a deterministic behavior) that produces this output (Figure 7).

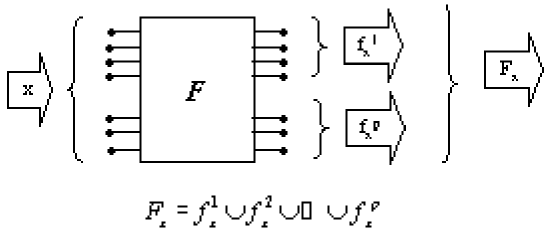

Fig. 7 Graphical description of a fully realizable function F

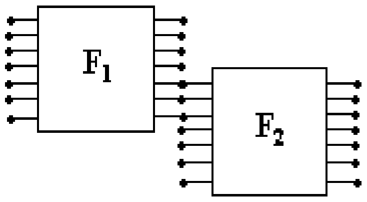

Fig. 8 Two composed components

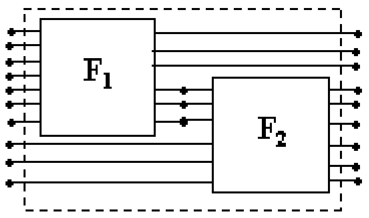

Fig. 9 Two composed components seen as one

(f): *The functionF is* **time-independent** *if the timing of the messages in the input streams does not influence the messages in the output streams but only their timing. [9]*

4.1.2. *Composition Rules.* Channels are assumed to have a type from a given set of types T assigned. A type is a name for a set of data elements. We consider the function *type* that assigns a corresponding type from the set T to each channel from the set C (type: $C \to T$).

Definition: *Two components that implement the functions F_1 and F_2, are said to be* **composed** *if output channels of the first component are used as input for the second one (See Figure 8).*

Remarks:

- Finally the two components can be seen as one (Figure 9).

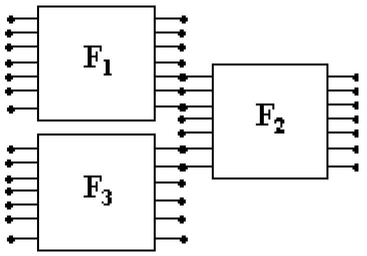

Fig. 10 Three composed components

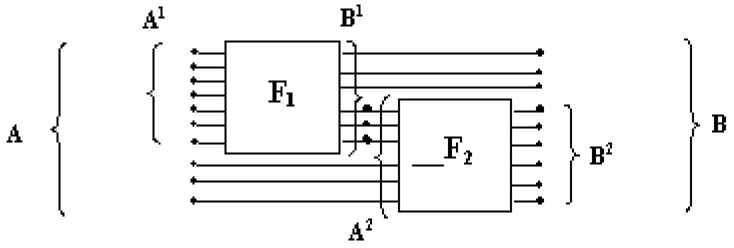

Fig. 11 A detailed view of two composed components

- The definition can be restated for more than two components (Figure 10).

We can say that the composition of two components is reduced to the existence of common elements in the sets B^1 and A^2, where B^1 represents the set of output channels of the first component A^2 represents the set of input channels of the second component. We can state that two components are composed if $B^1 \bigcap A^2 \neq \Phi$.

Condition. Two channels, one output B^1_n and one input A^2_m, can be used for the composition of the components if and only if *type* $(B^1_n) =$ *type*(A^2_m).

Consider two composed components (Figure 11).

The following properties can be stated:

a) $A = A^1 \bigcup (A^1 B^1)$, $B = B^2 \bigcup (B^1 A^2)$, $C = A \bigcup B$.

b) Considering that the processing time for the first component is τ_1 and for the second one τ_2, the processing time of the composed component is, $\tau = \tau_1 + \tau_2$, for the worst case

$$F = \begin{cases} F_1, & if \;\; A_x \subseteq AA^1 \;\; and \;\; B_{F(x)} \subseteq (BB^2); \\ F_2, & if \;\; A_x \subseteq (AA^1) \;\; and \;\; B_{F(x)} \subseteq B^2; \\ F_1 \circ F_2, & if \;\; AAA - x \subseteq A^1 \;\; and \;\; B_{F(x)} \subseteq B^2. \end{cases}$$

c) Using the notations from the first part of the article we describe the function that implements the composed component as follows:

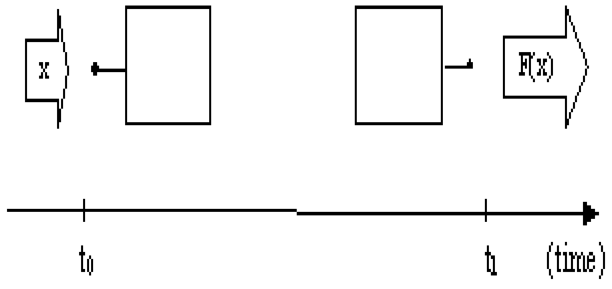

Fig. 12 A one input-one component

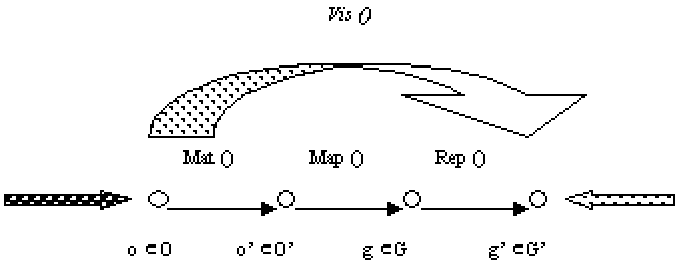

Fig. 13 Synthesis of the proposed visualization model

Remarks:

- $F_1^{\circ}F_2$ means that for an input x we get: $x \to F_1(x) \to F_2(F_1(x)) \equiv F(x)$
- The function described above does not include all cases. For example, the function is not defined for $A_x = A$ and $B_{F(x)} = B$. These cases are solved by decomposition, in order to fit into the given description.

4.2. **The Visualization Pipeline Using Components.** Consider the case of one input -one output component. The function F describing the component is considered to be time-independent *(definition (f))*. Two working steps can be revealed: at moment t_0 a message is introduced into the input channel and at moment t_1 the result is read from the output channel (Figure 12)

We considered that the implemented services are **consistent** *(definition (a))*. The component is **strictly causal** *(definition (c))* and **fully realizable** *(definition (e))* because the strictly causal function needed is F.

The reference model introduced to describe the visualization process contains three distinct steps (Figure 13).

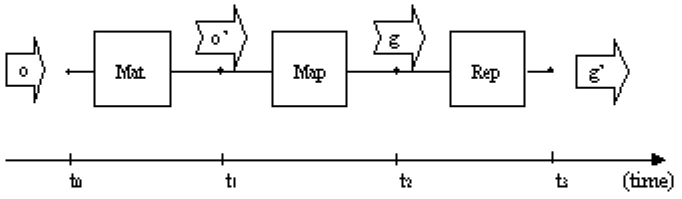

Fig. 14 A component-based visualization system

Suppose that each step/function is performed by an independent component. Then the visualization pipeline becomes the one in figure 14. The components should take into account the scientific visualization criteria, meaning that the *Mat(),Map(),Rep()* functions implemented by the components should be injective and *Vis()* should be bijective.

A further decomposition can be performed. Each component can be obtained from the composition of two or more components depending on the visualization process. A personalized visualization system is obtained at this level.

4.3. **Example: Visualization of natural gas reservoir using a component based visualization system.** In order to present a detailed component based visualization system we consider the example of a natural gas reservoir. A nonuniform data set obtained using drills is visualized. The data set is processed using two paths (Figure 15).

The filter component transforms the nonuniform input data into a uniform data set using the geological characteristics of the underground and the methane concentration. The filter also performs the sub-sampling of the data set. One of the paths includes the intersection of the data set with a plane. The generation of the isosurfaces represents another step of the modeling level.

The colors, the light, the position of the camera and other properties are set at the logical visualization level. A human-computer interface (HCI) allows the user to change the properties without interfering with the data flow.

Finally, two objects that can be rotated, translated and zoomed are obtained (Figure 16): one used for the dimensional exploration of the deposit and another for the visualization of the level lines. (The presented images were obtained using the VTK library, and the components were implemented using Tcl/Tk)

5. Conclusion

The paper addresses the issue of component-based scientific visualization systems. The visualization process validation criteria are presented in detail and a reference model for the visualization systems is proposed. The paper supports the idea that a component- based architecture solves most of the standard/commercial

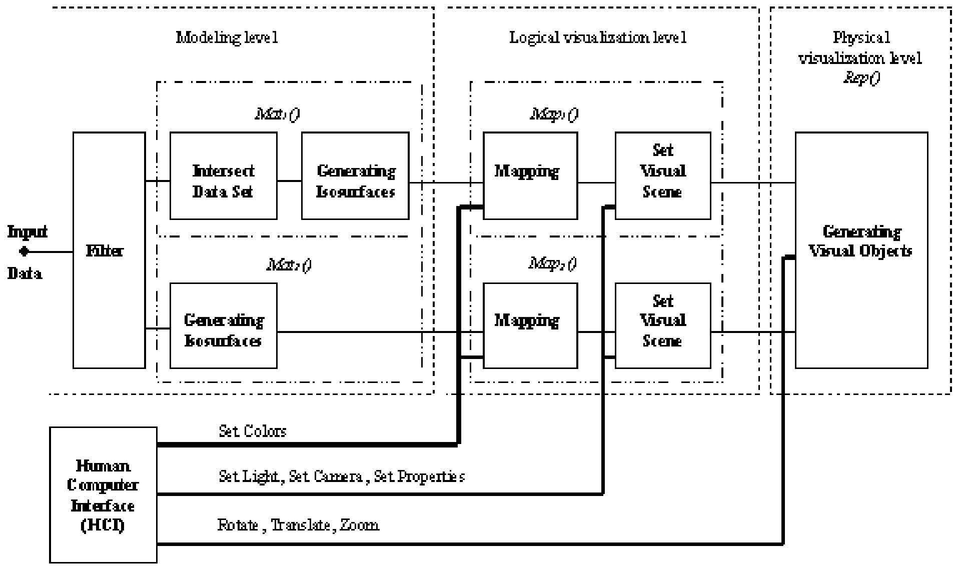

Fig. 15 The architecture of a custom made system used to visualize a natural gas reservoir

visualization system problems. It should be noted that the introduction of the component-based technology does not solve lead automatically to the wide spread of customizable scientific visualization systems unless a critical mass is reached. The theory is supported by a detailed example of a visualization application of a natural gas reservoir.

References

[1] Upson C., Faulhaber, Jr. T., Kamins D., Laidlau D., Schelgel D., Vroom J., Gurwitz R., van Dam A., *The Application Visualization System: A Computational Environment for Scientific Visualization*, Computer Graphics and Applications, vol. 9, nr.4, 1989.

[2] Radoiu D., Roman A., *Modelarea procesului de vizualizare, in Tehnologii Avansate Aplicatii in educatie*, Editura Universitatii Petru Maior, 1999, p.86-101

[3] Kaufman Arie, Nielson G., Rosenblum L. J., *The Visualization Revolution*, IEEE Computer Graphics, July 1993, p. 16-17.

[4] Hibbard W., Dyer C., Paul B., *A lattice Model for Data Display*, Proceedings of IEEE Visualization '94, 1994, pp. 310 - 317.

[5] Hibbard Williams L., Dyer Charles R., Brian E. Paul, *Towards a Systematic Analysis for Designing Visualizations*, Scientific Visualization, IEEE Computer Society, 1997, pp. 229 - 251.

[6] Radoiu D., *Scientific Visualization of Experimental Data*, Ph.D.Thesis, Universitatea Babes Bolyai, 1999.

Fig. 16 The visualization of the natural gas reservoir

[7] Radoiu D., *On Scientific Visualization Systems Design*, Studia, 1998.

[8] MacKinlay, *Automating the Design of Graphical Presentations of Relational Information*, ACM Transactions on Graphics, Vol.5, Nr.2 1986, pp.110-141.

[9] M.Broy, *Software Concept and Tools*, Springer-Verlag, 1998, 57-59.

[10] D. Radoiu, *Vizualizarea Stiintifica a Datelor Experimentale*, Editura Universitatii Petru Maior, 2000.

Petru Maior University of Tirgu Mures
E-mail address: `dradoiu@uttgm.ro`

Polytechnic University of Bucharest