APPLYING MODEL-VIEW-CONTROLLER IN GEOMETRIC MODELLING METHODOLOGY*

Alexander Penev, Dimcho Dimov, Vassil Vassilev

Abstract: *Article describes the union of architectural pattern MVC and five-layer architecture for geometric modelling. Main purpose is creation of high-modular interactive systems. The implementation includes different reusable object-oriented design patterns and shows a possible application of the architectural union.*

Key words: *Computer graphics, Geometric modelling, Methodology, MVC*

1. Introduction

Geometric modelling takes central place in modern Computer Graphics. Geometric modelling methodology defines how such system can be developed to be clear, extensible and stable.

Parallel to the implementation of classical schemata for interactive applications are established technologies for software engineering, which improve the system abstraction. These technologies include the application of high-reusable object-oriented design and architectural patterns. Such pattern is the so-called Model-View-Controller (MVC). It is used in systems, which should bring in loose coupling between the data, its visual representation and the methods of its manipulation.

According to the development purposes of the OpenF [5] system was made attempt for combination of the classical architectures for building geometric modelling systems and modern technologies for achieving higher abstraction. The result of this attempt is subject of present article.

2. Methodology for geometric modelling

In [1] is described methodology for geometric modelling. It is based on the logical separation of the system functionality for processing geometric information. Each functional group is called virtual machine or processor. Each virtual machine handles specific part of the graphic information as processes it and interacts with other virtual machines. Their purpose is to encapsulate relatively independent and similar operations. They communicate with each other via preliminary defined interface and eventually pass processed data. This hierarchy of virtual machines consists of:

- Display processor – via manipulating it, the system draws image,

* The work is partly funded by Fund Research of the University of Plovdiv under contract ИС-М-4/2008

using output devices and implements the events coming from the input devices. Important functionality is the picking (identification) object from the image. This processor is programmatically hardware system, which executes sequence of commands, called display file. The file represents an image (raster or vector);

- Geometric processor – main tasks are the geometric transformations of the image (such as translation, rotation, scale and projections), clipping operations, etc. Its major purpose is to implement the mathematical functions upon the visualized objects. It is possible to consist even of modules, which implement the necessary mathematical functions such as matrix multiplication etc. The basic idea of implementing such modules is reaching independence of the geometric processor and the use of different calculations with necessary precision;
- Structure processor – its functions are related generally in model structure maintenance, that is grouping elements of the image in subimages. It is clear that the structure processor depends on the model separation, i.e the model can be divided between processors or to be homogeneous structure. Consequently, the implementation of the structure processor is bounded up with the chosen application architecture;
- Semantic processor – transforms nongraphic models into image models. The processor maintains nongraphic information about the image elements (such as names, areas and prices) and its process algorithms. This processor makes five-layer architecture flexible and applicable for huge set of applications. Since every contemporary application even with nongraphic model needs visualization, the semantic processor can transform the nongraphic model into graphic and then to visualize it correctly;
- Dialog processor – it is the utility, which transforms user commands into operations over the image model and as result the image is changed. It implements the dialog logic between the end user and the computer. There are many possible approaches of implementation of the processor depending on the followed style of communication between the user and the application. Basic characteristic of the dialog processor is that it can be divided into subparts most commonly called tools. Such division corresponds with the contemporary tendency of event driven programming.

Precisely the combination of these virtual machines, their interaction and approach of architecture building define the methodology of the geometric modelling [1].

In [2] the idea of five-processor hierarchy was developed (see fig. 1). In this hierarchy, the processors are arranged according to the principles of the multilayer architectures. Each layer implements its func-

tionality with so-called services. A layer has its own internal state and uses only the services of the layer under it, (i.e. it is “unaware” about the layers above). Service activities are based on the internal state and the parameter passing from other layers. For instance, the action of the dialog processor can be illustrated via the formula 1:

$$(M', S') \leftarrow D(M, S, E) \quad (1)$$

where *M*, *S* and *E* are respectively Model, Internal State and Event. *M'* and *S'* are new Model and Internal State of the dialog. In other words, on occurring of definite event the dialog processor can act in response as to alter either model or the internal state, or both. User-system dialog can be accomplished on the foundation of this formula.

The image model can be separated:

- between layers;
- in one layer (the dialog);
- out of the processors.

The services are separated into two relatively independent groups – services from forward chain (SFC) and services from backward chain (SBC).

However, such view of the represented image means that each representation is new, more concrete model of a specific aspect of the abstract model. In addition, during converting should not be lost information, but only some aspects to be defined more accurately.

These layers (and the processors) are abstractions, i.e. depending on the concrete application they can be implemented even together. They can be missing or to be fully or partially implemented by the hardware (for instance, see OpenGL, which contains the most layers and it has many software and hardware implementations).

Architecture is applicable for the most modern interactive graphic applications such as 2D and 3D graphic editors and word processing systems.

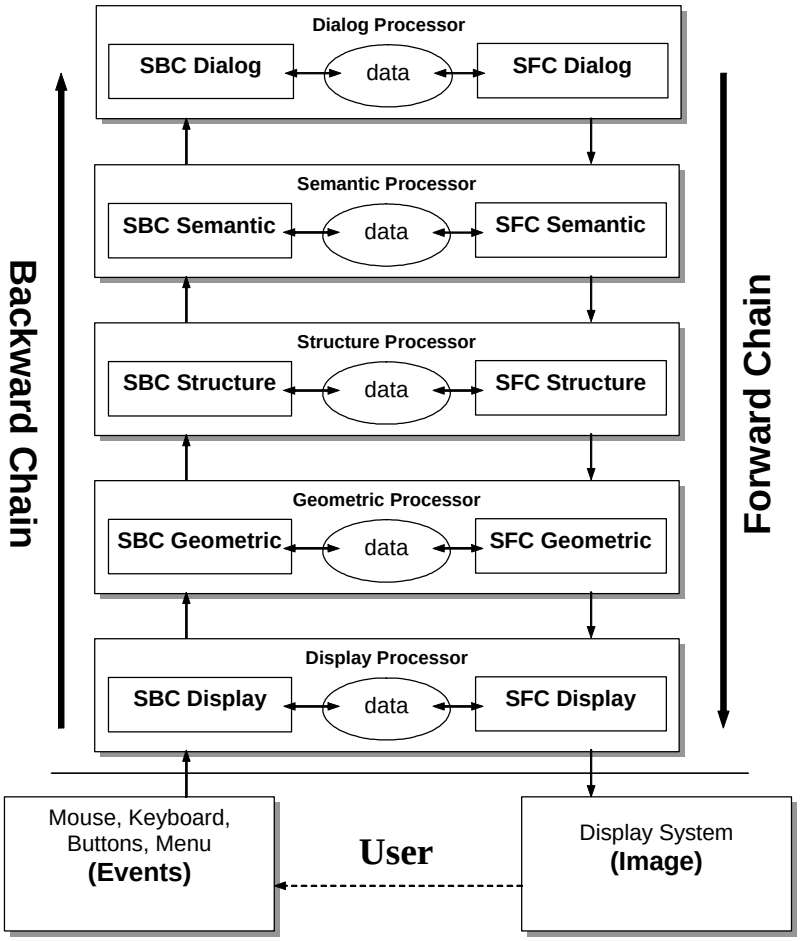

Figure 1. Geometric Modelling Methodology

3. MVC pattern

Model-View-Controller (MVC) is an architectural pattern for building applications (generally, user interfaces). It reveals its potential for first time in Smalltalk's [4] user-interface libraries. In the MVC paradigm the model, the user request and the visual feedback to the user are explicitly separated and handled by three classes; each specialized for its job. The MVC architecture consists of three types of objects (see fig. 2):

- The model manages the behaviour and data of the application domain;
- The view manages the graphical output of the display system, visualising the model;
- The controller interprets the input events from the user, changing the model or the view to change as appropriate.

As shown in [3], MVC is based on several more common design patterns, which purpose is to separate logically the system into three loose coupling segments.

Patterns Observer, Composition and Strategy [3] are responsible for establishing the connections in the MVC triad. Thus, the massive systems become modular, more flexible and easier for development and support.

The model is "unaware" of the existence of either the controller or the view and of its participation in an MVC triad [6]. Observer [3] takes the responsibility to notify the View when the model is changed. Controller implements the mechanism of changing the model via the user's requests.

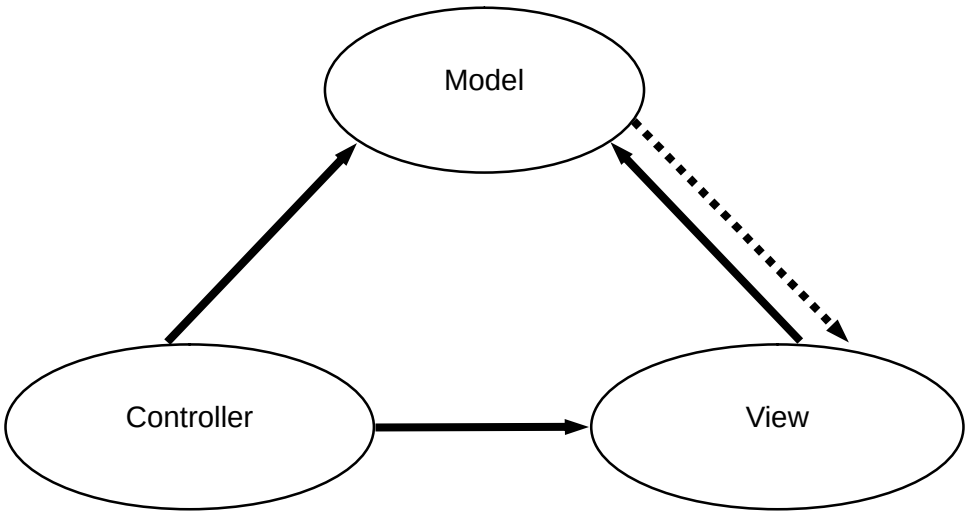

Figure 2. Model-View-Controller triad

Today MVC is widespread among the WEB applications, user-interfaces, frameworks, etc.

4. Implementation

Both approaches do not exclude each other. We have made a program implementation, which architecture obeys the schema (see fig. 3) and uses several popular design patterns.

In the described architecture, there are three kinds of data:

- Description of the geometric information (such as primitives, groups and characteristics) forms base model of the system;
- Common internal states of the layers (such as selection and markers) form optional helper models, accessible via the base model. If system elements do not use them, helpers are ignored. In fact, helper models are not part of the base model;
- Private internal states of the services (different activity modes

such as drag flags and key pressed) remain capsulated in the concrete implementation of the service.

The illustrated data separation leads to the storage of the application's geometric information at locations, which are uncontroversial to MVC, SBC and SBC.

The design pattern Composition [3] applied to the base model improves its flexibility and extensibility. For implementation of systems of this type, the design pattern Composition is most commonly used, because it allows the objects to be organized into hierarchies. This approach enables single object and compositions of objects to be treated equivalently. Using this technique the user can build complex sceneries, which consist of simple components. Likewise, components that are more complex can be formed via using grouping and the complex components can form even components that are even more complex. For processing these complicated primitives, the same mechanism is used. Obviously, if exists need of more sophisticated model structure can be used other more relevant conceptions can be used.

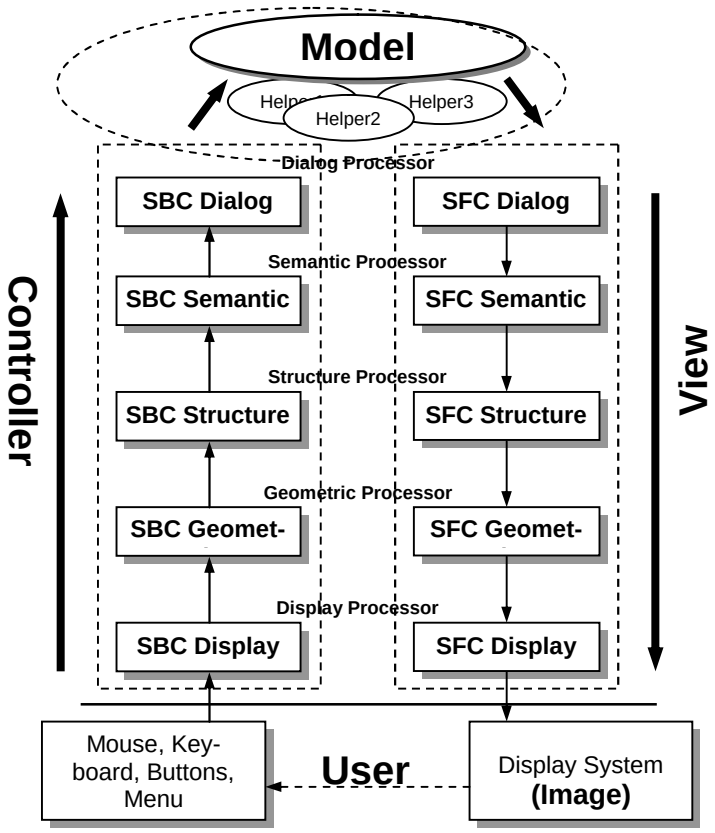

Figure 3. Five-layers applied on MVC

The five-class hierarchy distinguish the forward chain services, which forms the View segment from MVC. Separation has several advantages of major importance, such as:

- Modularity and saving the abstract functionality of each layer, capsulated in separate class;
- Service reuse from the previous layers (inheritance without change and inheritance with overriding);
- Access to all forward-chain services via creation of a single object, which is successor of the whole functionality (dialog processor);
- Extensible functionality.

There are two basic approaches for dividing the visualization part from the model. Each approach has its advantages and disadvantages.

Firstly, using the behavior design pattern Visitor [3] solves partially the problem as enables the visualization part to be “outside” the model. However, when extending the model or adding new visualization types, the class, which implements the Visitor pattern, has to be rewritten. Adding new primitives to the model should not affect visualization mechanism. Consequently this approach is weakly applicable for our purposes.

The second approach, which would not universally satisfy the problem, but it satisfies the special case in the geometric systems. Practically this approach represents emulation of double dispatch mechanism [3]. For saving the flexibility of the model, the visualization is separated into helper-classes called viewers. They help to move the visualization mechanism from the model towards the display processor, i.e. the view segment. Thus, the model could be extended without need of rewriting the visualization part each time. Besides, the viewers allow the existence of different ways in displaying the model (such as tree view and text). When visualization of element from the model is needed, the system looks for a concrete viewer in the table of the viewers and gives it the control. Adding a new type element or visualization requires only adding new viewer in the table.

The backward-chain services are also distinguished five-class hierarchy, which forms the Controller segment from MVC triad. Advantages are analogical to forward-chain’s advantages. Delegates (or so-called callback functions) take the responsibility of the event processing. Delegates allow an upper layer to declare processing of concrete events from a bottom layer. It occurs without knowledge of the bottom layer about the existence of an upper layer. Each layer can extend the information associated with an event and can pass it above.

Similar to viewers are used tools and Command patterns [3]. Tools are part from the dialog processor and give it specific new functionality, without need of change the controller classes. Each tool attaches itself to

required events and decides, according to its implementation, how to change tool state. During activity accomplishment, tools initiate appropriate change in the model.

The application of analogical approach (tools, viewers) for extending the capabilities of the forward and backward chain saves good logical arrangement. It allows the addition of new functionality without causing problems with the interaction between the virtual machines.

As it was described in the beginning, the dialog processor can be divided into subparts, which allows each subpart to be differentiated as self-dependent piece in the processor. Each subpart does not need to be "aware" of the existence of other, unless it uses functionality implemented in the other subpart. The so called subparts can be classified as:

- Core;
- Tools;
- Space of internal states.

All tools use the shared functionality of the processor, i.e. the core and eventually the space of the internal states.

The tools are "upgrade" of the dialog processor. They make the application interface user-friendly and logically clear. However, for implementing user-friendly interface is necessary the visualization of new images, which are not part of the model. These images are called markers. Markers belong to the helper part of the model and they manage the tools. Markers most commonly visualize the space of the internal states.

5. Results and Applications

The implementation represents interactive graphic editor, illustrating the idea of the architecture union. The successful application of patterns grants good flexibility and logical arrangement, which makes the described technique suitable for educational purposes.

The already made implementation (see sample screenshot at fig. 4) uses approved concepts for organizing of the user interface in the modern graphic editors. There is multi-document user interface (MDI), which simplifies the user interaction. The colour palette, action history and tools palette can be docked flexibly among the visible region. Integrated MS Agent can give advice, during work, what action should be taken.

A significant advantage of vvGraphic is that it can be extended easily. Adding a new primitive, new tool or other functionality does not need rewriting of the existent classes, which practically saves the loose coupling of the system.

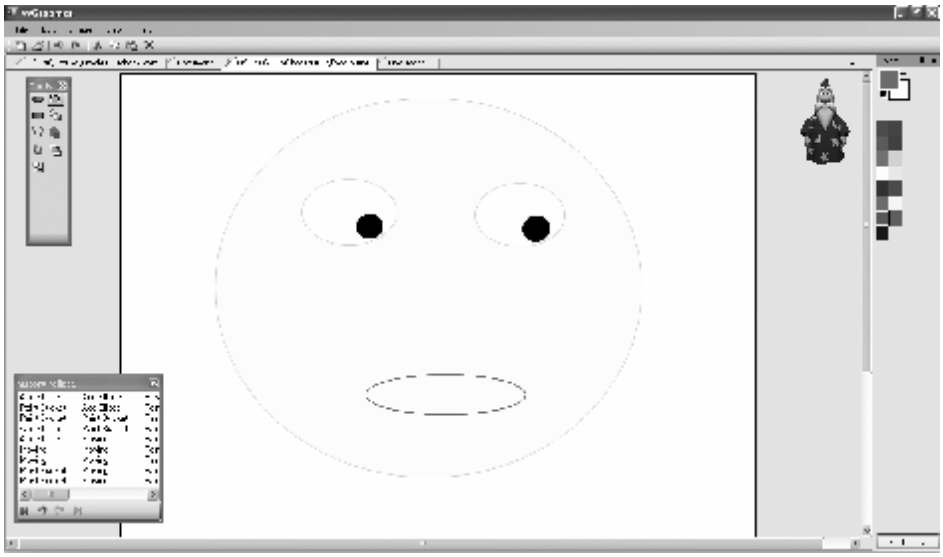

Figure 4. Example screenshot of vvGraphics geometric system

The development results lay good foundation for their application in OpenF-based hybrid system for geometric modelling – another step in the implementation of the OpenF educational examples.

The application of the MVC architecture contributes for the transformation of desktop geometric modelling systems into web based.

6. Conclusion

The conceptual architectures (fig. 1 and fig. 3) were implemented in “Computer Graphics” course, taught by the authors in Faculty of Mathematics and Informatics at University of Plovdiv “Paisii Hilendarski”. Observability and modularity of the architecture help the students to get familiar with the basic principles of the geometric modelling methodology. Simultaneously was used contemporary programming techniques and object-oriented design of applications. This makes the approach widely applicable for educational purposes.

Undoubtedly, the combination of MVC and the five layers provides many advantages:

- Better modularity;
- Better simplicity and clearness in development of powerful geometric modelling systems;
- The approach allows and simplifies plug-in architectures;
- Better model representation, visualization and manipulation;
- The system elements remain more “loose coupling” and easily reusable.

A problem is lower performance (generally in visualization) as consequence of the better modularity, flexibility and openness.

Future progress in the research should be in eliminating these problems, i.e. achieving better performance in visualization. In fact, the problem with lower performance in high modular (with good design) or high-parameterized object-oriented systems is widespread. This gives the idea in research for the use of automate optimization in object-oriented applications. The optimization should be made in the executable code and should not affect on the good design and openness. This is extremely wide domain, in which application could be automated problem-oriented optimizations.

Another direction for future progress of the thematic is examination the use of Hierarchical Model-View-Controller (HMVC) and Service Oriented Architecture (SOA) for geometric modelling.

References

1. **Dimov, D.** Computer Graphics (in Bulgarian), PUP, 1999, Plovdiv
2. **Dimov, D., Penev, A.** Computer Graphics Guidebook (in Bulgarian), 2002, Plovdiv
3. **Erich, Gamma, Richard Helm, Ralph Johnson, John Vlissides.** „Design patterns: elements of reusable object-oriented software", Addison-Wesley Longman Publishing Co. Inc., 1995, Boston, MA
4. **Krasner, G., Pope S.** A cookbook for using the model-view controller user interface paradigm in Smalltalk-80", Journal of Object-Oriented Programming, August/September, (1988), 26-49
5. **Penev, A., Dimov, D., Kralchev, D.** Open hybrid system for geometrical modeling", 17th International conference SAER-2003, vol. 1, (2003), 131-135
6. http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

Authors

Assist. Prof. Alexander Plamenov Penev
FMI, University of Plovdiv „Paisii Hilendarski"
236 Bulgaria blvd, Plovdiv, BULGARIA
apenev@uni-plovdiv.bg
Assoc. Prof. PhD Dimcho Stoikov Dimov
FMI, University of Plovdiv „Paisii Hilendarski"
236 Bulgaria blvd, Plovdiv, BULGARIA
dimcho_dimov@abv.bg
Vassil Georgiev Vassilev
FMI, University of Plovdiv „Paisii Hilendarski"
236 Bulgaria blvd, Plovdiv, BULGARIA
vassil_vassilev@hotmail.com