A Method for Smart Graphics in the Web

Thorsten D. Mahler, Stefan A. Fiedler, and Michael Weber

University of Ulm
Department of Media Informatics
Ulm, Germany
{mahler | fiedler | weber}@informatik.uni-ulm.de

Abstract. Shifting the attention from simple static depicting to dynamical generation of visualizations smart graphics can be seen as the next step in the evolution of presentation techniques. A new field of interest for these techniques is the Web which arises the question of how to adopt this new pattern. Based on the Model-View-Controller (MVC) paradigm we developed a generic architecture that allows for user interaction, component separation, adaptivity to environment and user needs and dynamic presentations. By introducing a client side presentational logic component the graphic evolves from a simple presentation to an intelligent visualization that valuates data and presents only the result relevant to the user. We consequently introduce the possibility of completely changing the whole presentation including the user interface by adaptation of the valuation basis. Because this is done by the graphic autonomously the server does not have to be contacted. In order to demonstrate our proposed conceptual approach, we implemented a prototype system by using the techniques available in a Web context.

1 Introduction

Illustrations, images, graphics as they are known today are static presentations used to show concepts, illustrate theories, and present results. In contrast, computers being used as medium to show illustrations provide possibilities that are far beyond this static understanding of graphics.

In this paper we present an approach that utilizes smart graphics for the Web. At first we present a short analysis of the features of smart graphics systems to define the task our conception has to meet. After that we shortly introduce two referenced models and present our approach for a distributed smart graphics system followed by a short description of our prove of concept implementation.

2 Smart Graphics Features

As we focus on an overall conception for smart graphics systems in the Web we firstly present an exemplary selection of smart graphics systems to point out its specialties and advantages included in our approach.

Visual Map: Visual Map [2] is a project from the area of 3D Geographic Information Systems (GIS). Its goal is to provide the user with an overview on his location by creating a map optimized for his needs. The system is adaptive as it shows the part of the map just needed. It supports orientation by reducing unnecessary information and introducing orientation aids (e.g. landmarks). Furthermore Visual Map is adaptive to hard- and software constraints as well. To achieve this behavior the system has to weigh its input data and to connect it to semantic knowledge about the graphical objects.

WIP/PPP: Primarily WIP/PPP [1] is an authoring system based on a knowledge base which supports the automatic production of multimedia presentations. The modeled life-like characters show, explain, and comment processes and illustrations. The WIP/PPP multimedia authoring system produces an applet which contains all elements and logic for the presentation. WIP/PPP even uses the Web but only to transport the resulting application to the host system - its focus definitely lies on the intelligent interface.

ZoomIllustrator/TextIllustrator: The ZoomIllustrator [7] project and its successor TextIllustrator [4] allow the user to interactively experience 3D-Illustrations. He can explore details of the graphical representation and the system simultaneously provides textual information of the focused object. The textual representation of the portion needed is expanded whenever a more detailed version is required. So this system has to be able to keep both its presentations, graphical and textual, consistent whenever an interaction takes place.

Summarizing these system demonstrate the adaptivity of smart graphics to changes: They are able to recognize requirements and constraints, adapt to users needs and preferences, and react accordingly. They use different approaches to improve the interface; they including domain knowledge in the systems themselves to enable the systems to evaluate semantic knowledge in order to weigh data and react to user interaction. As these are the tasks our method has to meet in the following a smart graphic is defined as a system which is capable of autonomously adapting its presentation to constraints, of valuating changes, and of dynamically updating its visualization accordingly.

3 Reference Models

From a conceptual point of view our work is influenced by two paradigms which are explained in the sequel.

Model-View-Controller: The basic and widely accepted paradigm to model user interaction is the Model-View-Controller (MVC) paradigm [6]. MVC is based on the idea of object orientation and therefore divides an interactive application into three components:

1. A model component which includes the application domain state and behavior, handling all changes of the actual data,
2. A view component responsible for data layout and visualization, and
3. A controller component managing user input and device interaction.

This paradigm enables interactive systems to use different views of the same model at the same time and to keep them synchronously updated. It identifies the functional parts and divides them into independent parts. Therefore, it is a valuable concept for interactive systems. However, the realization of the paradigm is problematic, since especially view and controller have to cooperate closely. In consequence, they can only be separated up to a certain degree [8].

Layout-Content-Logic: The second paradigm our system is based on is the Layout-Content-Logic (LCL) paradigm [5]. This paradigm originates from the area of web engineering and was introduced to solve one of the biggest problems in this field: The commingling of layout, content, and logic.

The mix of content and layout widely has been recognized as disadvantage. The problem of intermingling logic with content or layout is the new aspect that is addressed in this paradigm. Logic in the LCL paradigm refers to the parts responsible for dynamic content creation.

Logic in MVC and LCL: In the MVC paradigm the functional core of the application is the logic contained in the model. In contrast to that the logic component of the LCL paradigm encapsulates the parts responsible for the dynamic content production. So while the LCL logic is purely presentational, the logic contained in the model component of the MVC paradigm is applicational. This differentiation is an important point for smart graphics because they incorporate both: They are interactive systems and use dynamic content creation techniques as they evaluate changes and adapt their presentation likewise.

4 Distributed Smart Graphics

As we are interested in modeling an interactive system our concept will be based on the presented MVC paradigm. However, MVC does not distinguish between applicational and presentational logic. With smart graphics, as stated in Sect. 2, the presentation of the data itself is generated. This functionality neither is part of the MVC model nor is part of the MVC view. We bridge this gap by introducing the presentational logic component that receives the data from the model, processes the presentation data, and passes the newly processed data to the view (see Fig. 1). The separation of applicational and presentational logic allows to procedure the actual presentation as the last step in the chain of data processing and therefore such applications are highly flexible and able to react immediately to interaction.

Fig. 2 depicts the distribution of the identified components, which are explained in the following:

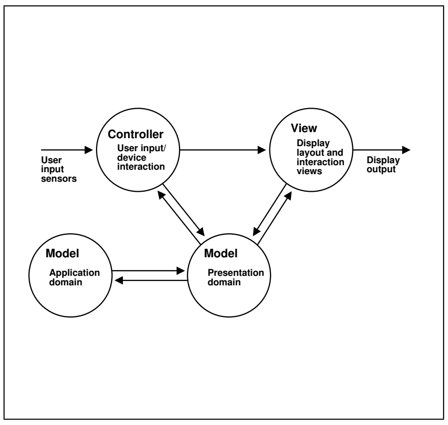


Fig. 1. Enhancement of the MVC architecture for smart graphics.

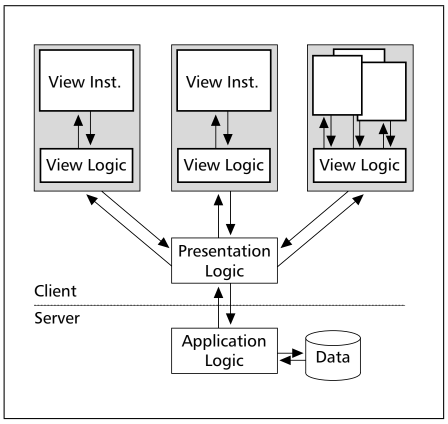


Fig. 2. Architecture and distribution on client and server.

The Server Side Applicational Logic: The applicational logic is the functional core of the application. It is the only component with direct access to the actual data. As it resides on server side the data can be supervised, protected and shielded from unauthorized access.

Client Side Presentational Logic: The presentational logic acts as the model in the MVC paradigm for the other components. Its task is the communication with the applicational logic to get the data required for presentation and to valuate them afterwards. By adapting the valuation basis the whole presentation can completely be changed. The result of the valuation is passed to the views and an update is initiated. The presentational logic can handle any number of views.

View: The view in this concept is divided into view logic and view instance:

The view logic contains the controller of the MVC paradigm and thus is capable of supporting user interaction. We already know that view and controller have to be tightly coupled. To nevertheless achieve a high degree of independence, we propose to additionally add all logic the view requires to this component. This bears the advantage that the view instance is only a realization of the requested presentation.

The view instance is the realization of the visualization. All visual parts are subsumed in this component. It depends on the view logic, because it holds all the code needed to make this component dynamic. Therefore the view instance and the view logic communicate whenever a change is made, i.e. by changes in the model passed through the presentational logic to the view logic or by changes caused by user interaction with the view.

4.1 Advantages of the Architecture

Adaptation to the user: The possibility of building more than one view and the configurability of the views allows high adaptation to user preferences and needs. For instance a view can be highly configurable, so that every user is able to adapt the view in any way he likes. It is possible to provide a certain number of preset views in order to increase and simplify usability.

Adaptation to the environment: By decoupling the presentational logic from the rest of the system it is also possible to react to different environments and host configurations. The presentational logic can test hardware and software conditions of the host system and accordingly load a suitable view.

Dynamic views: Besides the possibility of static views that just present data, views as well can be dynamic. Here dynamic refers to how view instances are created, i.e. they are produced by a logic part integrated into the view.

Adaptation to presentational tasks: Collaboration of server side and client side logic can be established in order to react to presentational tasks not even known, when the application is loaded. Changes in the underlying model can be made that will have to be visualized. In this case, the presentational logic can request a new visualization from the server and so is able to update the current views.

Application development: A parallel development of view logic and view instances by software developers and e.g. visual artists is possible with this approach. Thus the software developer can focus on programming the logical parts leaving the task of designing the view instances to professional designers that are skilled in the use of special tools for exactly this purpose.

5 Example Implementation

After this conceptional view of smart graphics on the Web we will present an example in order to illustrate the theoretical concepts. On the one hand the reaction to dynamic changes will be demonstrated and we will show the use of different views.

As a demonstrational we chose a spacecraft as technical system and as task the supervision of the life support system. Our implementation constantly receives sensor data from the whole spaceship, which has to be valuated. Finally the visualization has to be presented by the smart graphic.

5.1 Techniques Used in our Approach

In order to realize our distributed architecture on a wide range of systems, we had to decide how data can be modeled and exchanged between the components and which language is most suitable for programming the logic parts of the smart graphic. Since the implementation should be usable in the Web, it is obvious to decide for standardized web formats and techniques.

For the task of storing and exchanging our different kind of data we use XML, because it has been developed exactly for this purpose. In addition it is a format that is easy to manipulate and open for new standards that are not even yet developed.

As graphic format we chose SVG which is based on XML and therefore profits from many tools available for parsing and processing XML data. Since we opt a web application the choice of the programming language on the client side is restricted by the web browser. Thus we chose ECMAScript over third party solutions like Java applets or plugins. ECMAScript allows the easy integration of all other used formats by standardized procedures via DOM.

5.2 The Presentational Logic

The presentational logic is the central component on the client side. It periodically requests the up-to-date model data from the applicational logic and receives an XML file in response. Thereupon the data is valuated by checking the tolerable ranges denoted in our valuation base. The affected views are informed and an update is triggered using the Subscriber-Notifier [3] concept. It has to be noted that in this stage we are on a purely abstract level. The actual visualization and thus the presentation has to be done by the view.

5.3 The View

The view forms the interface of the application to the user. Its task is the presentation of information to the user and reading interaction from the user.

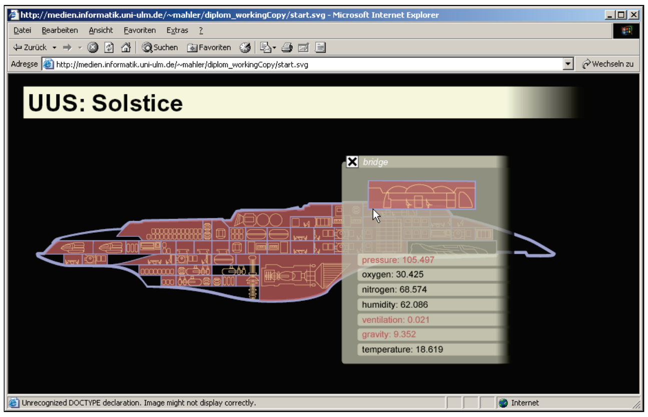
The view logic contains all code to react to changes made by the user through interaction and changes initiated by the presentational logic. Every view instance has exactly one view logic, whereas one view logic component can act for more than one view instance.

The view instance is the part that finally presents a visualization to the user. Conceptually there is more than one way how a view instance is formed. We examined three different kinds of views:

Initial presentation of the view instance: The view contains the displayable view instance. The view logic as a consequence is smaller because it only has to carry out changes on properties of the view instance, while the view instance itself remains unchanged to a large extent. We used this approach to present an initial cutaway of the spaceship to allow a rough diagnosis at a glance. (see Fig. 3, spaceship in background). In our example the view logic reacts to notifications of the presentational logic by changing the background color of the affected room.

View instance by template: The second approach puts more emphasis on the view logic. Nonetheless an initial view instance as static part exists. This initial view instance is not simply shown, but rather used as a template that dynamically is filled with data. This approach has the advantage of being able to use simple and fast tools for template production. We use this method for a second class of views, the inspection windows (see Fig. 3).

Complete generation of the view instance: The final possibility to obtain a presentation is the complete generation of the view instance by the view logic. Initially, the view does not contain a view instance at all. In fact it is generated on demand. This is the most flexible way considering that the view logic has full control of the view instance.



http://medien.informatik.uni-ulm.de/~mahler/diplom_workingCopy/start.svg - Microsoft Internet Explorer

http://medien.informatik.uni-ulm.de/~mahler/diplom_workingCopy/start.svg

pressure: 105.497
oxygen: 30.425
nitrogen: 68.574
humidity: 62.086
ventilation: 0.021
gravity: 9.352
temperature: 18.619

Unrecognized DOCTYPE declaration. Image might not display correctly.

Fig. 3. Our application showing presentations of two different views.

6 Conclusion

Based on the knowledge in the field of web engineering we presented an approach of how to realize and use smart graphics on the Web. Our conception was expected to model a system being able to independently adapt its presentation to conditions, to valuate changes, and to update its visualization accordingly.

Updates are triggered by changing model data and user interaction. Therefore we suggest a conception based on the MVC paradigm and the LCL concept that accomplishes an effective separation of the applicational logic and the view by introducing the presentational logic component. This new component valuates the applicational data before the presentation by the view. Moreover it decides on appropriate visualizations according to environment and user preferences. Conceptually it provides a separation of the functional parts: The sensitive data remains on the server, where it can be shielded from unauthorized access whereas the presentational components reside on the client. This reduces the web traffic significantly because conventional server based systems have to transfer the whole presentation with every user action. In our approach we transfer in the first step logic together with the presentation to the client, which subsequently can handle changes by itself.

On client side the view only depends on the presentational logic. The views task is the presentation of the valuated data. It is divided into view logic and view instance. The view logic contains the code necessary for operating the view instance. The code includes functions to react to changes in order to dynamically update properties in the view instance and to react to user interaction.

The introduction of smart graphics provides a fortification of the presentation by combining the simple presentation with logic. This leads to a more dynamic and flexible presentation. However, that does not mean that the visualization simply presents dynamic data, it presents a valuated visualization based on the preprocessing of applicational data by the presentational logic. Furthermore a faster development and better reusability can be achieved by the separation of the components.

References

1. E. André, J. Müller, and T. Rist. WIP/PPP: Knowledge-Based Methods For Automated Multimedia Authoring. In *Proceedings of EUROMEDIA96*, pages 95–102. PUB-SCS, 1996.
2. V. Coors. Feature-preserving Simplification in Web-based 3D-GIS. In *Proceedings of the International Symposium on Smart Graphics 2001*, pages 22–27. ACM Press, 2001.
3. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, 11th edition, 1997.
4. K. Hartmann, S. Schlechtweg, R. Helbing, and T. Strothotte. Knowledge-Supported Graphical Illustration of Texts. In S. L. Maria De Marsico and E. Panizzi, editors, *Proceedings of Working Conference on Advanced Visual Interfaces*, pages 300–307. ACM Press, 2002.
5. C. Kerer and E. Kirda. Layout, Content and Logic Separation in Web Engineering. In *LNCS 2016 - Web Engineering: Software Engineering and Web Application Development*, pages 135–147. Springer Verlag, 2000.
6. G. E. Krasner and S. T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1(3):26–49, Aug./Sept. 1988.
7. B. Preim, A. Raab, and T. Strothotte. Coherent Zooming of Illustrations with 3D-Graphics and Text. In W. A. Davis, M. Mantei, and R. V. Klassen, editors, *Proceedings of Graphics Interface '97*, pages 105–113. Canadian Information Processing Society, 1997.
8. Y.-P. Shan. MoDE: A UIMS for Smalltalk. In *Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications (ECOOP/OOPSLA)*, pages 258–268. ACM Press, 1990.