Adaptable Visualization Service: through Uniformity towards Sustainability

Tomáš Dymáček[1] and Petra Hocová[2] and Miroslav Kintr[3]

[1] Mycroft Mind a.s., Lidicka 28, 602 Brno, Czech Republic
[2] Knowledge and Information Robots Laboratory,Faculty of Informatics, Masaryk University, Botanicka 68a, 602 00 Brno, Czech Republic
[3] Institute of Computer Science, Masaryk University, Botanicka 68a, 602 00 Brno, Czech Republic
dym@mycroftmind.com,petra.hocova@fi.muni.cz,mirek.kintr@mail.muni.cz
http://kirlab.fi.muni.cz/en:homepage

Abstract. This paper presents a design of Adaptable Visualization Service which supports automatic generation of visualizations and GUIs. Adaptable Visualization Service is able to perform a wide range of visualization tasks. The advantage of presented service lays in its ability to combine various visualization methods and easiness of adding new ones. When adding a new visualization method the degree of changes made in Adaptable Visualization Service itself needs to be minimized. We assert that the desired minimization of changes can be reached thanks to uniformity of the proposed solution, adoption and extension of Model-View-Controller pattern, language independency of solution and division of service into three separate layers: language layer, visualization methods layer and transformation layer. Then the necessary changes can be limited to a small and isolated part of the visualization methods layer. We state that current solutions dealing with automatic GUI generation are difficult and costly to extend.

Key words: model-based user interface development, conceptual model, GUI design, Model-View-Controller

1 Introduction

Nowadays the information society suffers from information overload. Users looking for their answers in information systems can easily get lost, confused and disoriented. There are various kinds of information and these can be found in various data sources. After users input their queries into any information system there are usually two phases to be proceeded; the data retrieval and the data visualization, i. e. first appropriate data need to be searched out and obtained by the information system and second results of the search need to be visualized in a such way to be easily readable and understandable by the user. Every visualization method brings its own added value in expressing the information within the data. For given set of data the adequacy of used visualization method differs

within the scope of effective presentation. In some cases visualizing information as a list could be more appropriate than in the graph representation and vice versa. Described situation is more or less the same in most of the domains (i.e. information overload, need of effective visualization).

To explain it more properly some examples from the network security field follows. The domain of the network security field was chosen because of authors work in development of a system which provides support in this domain (e.g. [16]). The WhoIs information (information about internet domain details) might be visualized in a form because of its simple structure and its qualitative characteristics. A topology of a guarded network would be preferably visualized as a graph. Another example here is information about particular flows (i.e. record of network communication between two IP addresses). It is a quantitative information and could be visualized either as a list or a statistical diagram.

What particular visualization method for displaying certain kind of information is going to be used in real situation is specified in cooperation with a (potential) user. Every professional working above the data from the network security domain need to see the data in different visualization structure. Within the data on one hand security managers are looking for attacks done in/towards the guarded network and on the other hand network administrators are more interested in amount of transferred data. Every profession requires different highlighting of information at the first look.

Also requirements on the visualization service were evolving through time and more visualization methods were demanded to satisfy a system operator (i.e. security manager or network administrator) needs. The operator preferences were changing–a wider set of information needed to be computed from the network traffic data (process of data retrieval) and therefore a visualization of the new set had to be adjusted to the new kind of content. E.g. WhoIs information should be visualized together with information from databases of known attacks and the information about the physical location of devices.

To be able to visualize the all complexity of incoming information into the visualization service of the information system a new visualization technique so-called Nested Visualization was developed. The Nested Visualization approach is based on the principle of combining various visualization techniques. Is means that any of implemented visualization methods (a table, a mind map, a form) can be inserted into any element of implemented visualization method (a cell of the table, a node of the mind map, a cell of the form). Precisely, this approach allows to display the information within the whole complexity, in one window and by adequate visualization method. It also means that visualization of the result of some query can be adapted to the content of the result itself. The sample of such output can be seen the figure 1, where the mind map technique and the form visualization is combined in one workspace.

Moreover, to fulfill continuously changing requirements on the visualization service of the system and to be able easily handle the Nested Visualization approach a new architecture for the visualization service was developed and implemented, it is so-called Adaptable Visualization Service (AVS). The core of

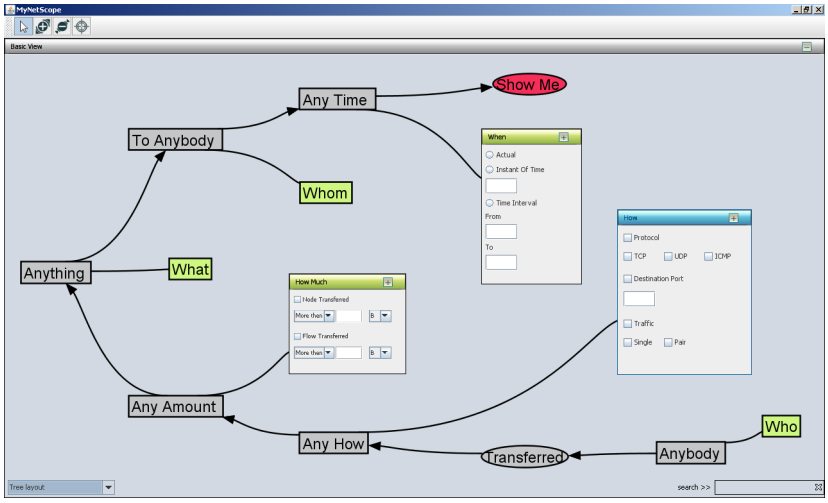

Fig. 1. The sample Nested Visualization, an output of AVS

the AVS serves as a platform into which desired visualization techniques can be plugged and then combined in by the Nested Visualization approach. This paper focus on important aspects of conceptual design and architecture of the AVS.

The regular job of a common visualization service is to render visualization of data according to a description, the description is prepared by other parts of the system (in process of data retrieval). Every visualization service accepts input in some defined input format, then it transforms this input into a GUI instance assembled from components provided by the visualization service. When adding a new visualization method the visualization service itself has to usually be reprogrammed. The phase of implementation of a new visualization method into existing visualization service must be performed quickly. The delay between identifying new requirements and their satisfaction must be as short as possible. Unfortunately, reprogramming the service and implementing new methods into it is very time consuming. Ideal situation for such task would be when the changes necessary for adding a new visualization method would be well defined, isolated and small. And the changes made within the visualization service because of adding new method would not propagate through the whole visualization service.

When the authors were designing the AVS they kept in their minds mentioned problems. Therefore they focused on narrowing the interfacial area between the core of the AVS platform and a possible new visualization method. To isolate and define such parts of the service that are necessarily going to be changed when adding new method the Adaptable Visualization Service is divided into three layers:

i) *language layer* – it is about the language (or format) of the input, specifies the elements of the language accepted and understood by other layers, it

contains and is built upon the conceptual model of abstract entities of Nested Visualization and other entities of the AVS, existence of this layer is important for the ability of the AVS to generate the GUI automatically from formalized description (language) (see section 2);
ii) *visualization methods layer* – the set of components implementing the visualization methods from which the resulting GUI can be built (these are either components or libraries of third parties or components implemented by our team); this layer also contains some more abstract entities, their role is to serve as uniform interface to the transformation layer;
iii) *transformation layer* – the engine for creating the GUI from the available components (from visualization methods layer) according to given input (coming from the language layer), this layer can be seen as a core of the visualization platform, but of course both other layers embody some elements of the platform too.

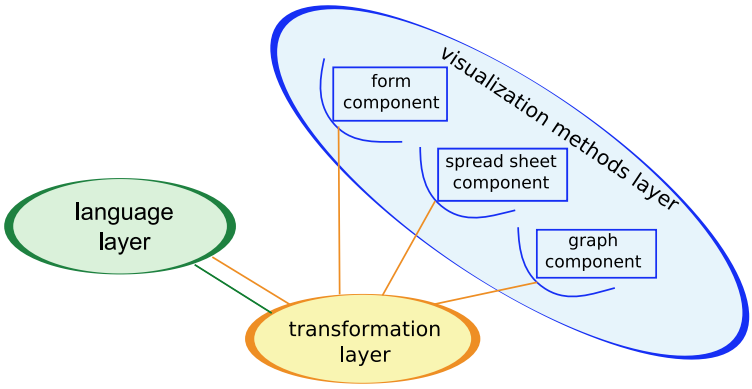

Fig. 2. The three layers of the AVS

The scheme of the layers is shown on the figure 2. The visualization methods layer offers the uniform interface across all implemented visualization methods to the transformation layer and transformation layer works with absence of knowledge above what visualization method is actually working. It means when implementing (adding) new visualization method into the AVS, it is necessary to work only within the visualization methods layer. The new visualization method is wrapped to fit with the interface of transformation layer properly. In a result such decomposition of the visualization service brings saving of time for implementation and limit the percentage of new bugs.

The duration of the implementation phase is shortened even more. It comes from the following concept. A set of methods implementing every particular visualization component (in sense of component itself, not the implementation into the visualization service) can be divided into three categories: i) data layer

- methods handling the data (i.e. the content of the visualization), ii) form layer - methods setting the form of the visualization (e.g. table), and iii) behaviour layer - methods determining the behaviour of the visualization component (e.g. sorting columns). The methods handling the data and the methods determining the behaviour of the elements are quite similar for any visualization component. So when implementing a new method into the AVS, it is possible to deal with these methods quite similarly across all visualization methods. Only the implementation of the methods setting the form of the visualization is necessary when adding the new visualization method into the AVS.

Described classification of the methods is known as the Model–View–Controller design pattern (MVC). The Model represents the data layer, the View is the form layer and Controller plays the role of the behaviour layer. In these days, there are implementations of many visualization methods available in the form of program libraries written in various programming languages. It is advantageous to use these libraries instead of programming it. As the desired visualization method might not be implemented in our preferred programming language the visualization service must be language independent. The problem of language independency is solved by the concept of three layers (language-transformation-visualization methods) where the added visualization is wrapped within the visualization methods layer.

2 Related Work

There exist several approaches which deal with automated GUI generation from formalized description.

XUL – XML User Interface Language is an XML user interface markup language developed by the Mozilla project for use in its cross-platform applications, such as Firefox. The only complete implementation of XUL is the Gecko layout engine. XUL relies on multiple existing web standards and technologies, including CSS, JavaScript and DOM [15].

JFCML – JFC/Swing XML Markup Language is a markup language for Java, which specifically targets the creation of AWT/Swing Graphical User Interfaces. More formally, JFCML is an XML User Interface Language (XUL) for Java. JFCML has been designed to be easy to use, yet powerful enough to write a complete application [11].

XAML – Extensible Application Markup Language is a declarative XML-based language used to initialize structured values and objects. XAML is used extensively in the .NET Framework 3.0 technologies, particularly in Windows Presentation Foundation (WPF). It is used as a user interface markup language to define UI elements, data binding, eventing and other features [6].

UsiXML – USer Interface eXtensible Markup Language is a XML-compliant markup language that describes the UI for multiple contexts of use. UsiXML supports platform and device independence [23].

UIML – User Interface Markup Language is an XML language for defining user interfaces. It allows to describe the user interface in declarative terms (i.e.

as text) and abstract it. Abstraction means that it is not necessary to specify exactly how the user interface is going to look but what elements are going to be displayed and their behaviour [18].

Let us have a look at how these approaches are prepared for extension by new visualization methods.The problem of solutions built on XUL, XAML, JFCML and UsiXML lies in their non-uniformity. Each GUI element described by its XML definition has its own XML tag in these solutions. To add a new GUI element into existing solution requires: i) to add a new tag description to their DTD – i.e. modification of the language layer, ii) to adapt the interpreter – i.e. transformation layer. Especially point i) is a problem for sustainability and later compatibility (e.g. when local user defines his own extension of DTD).

UIML is the most promising solution from our point of view. But its drawback lies in its lack of support for the MVC pattern. Using this approach would require to implement the whole visualization method from scratch without the possibility to reuse models and controllers from other visualization methods.

As none of the approaches known to us suits the requirement of easy extensionability AVS has been developed.

3 AVS – The Adaptable Visualization Service

AVS is an implementation of a visualization service which meets the requirement of extensibility.

3.1 Architecture

AVS consists of three layers: i) language layer, ii) visualization methods layer iii) transformation layer. The language layer provides an apparatus for a conceptual description of GUI, the visualization methods layer provides components from which a GUI can be built and the transformation layer can transform given input (GUI instance description) into the output (instance of GUI).

Language Layer: The language layer plays the role of the conceptual background of the whole AVS and it is described by the conceptual model (see Fig. 3 - in a Entity-Relationship diagram notation). The entities from the conceptual model are also elements of the description language. As it is visible from the model, they are all on quite high level of abstraction.

This helps to the transformation layer works uniformly with any visualization method without knowing unnecessary details about it. From this approach the main advantage of the language layer results- it is not necessary to change the conceptual model if one wants to add a new visualization method implementation, only a new instance of the existing model entity called Visualization has to be inserted (from the viewpoint of the mention layer). This solution is sustainable because the structure of the language layer does not change over time.

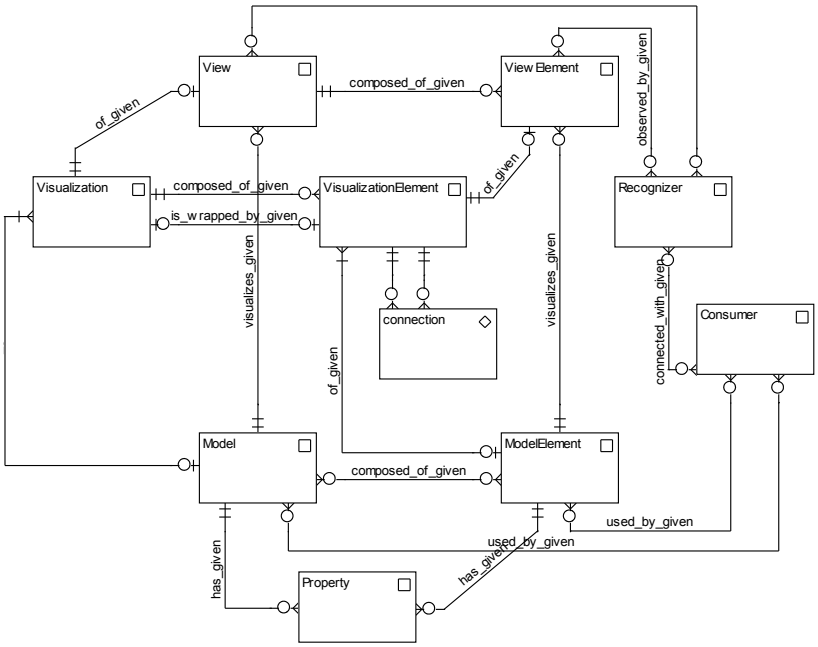

Fig. 3. The conceptual model of input layer

Let us have a look at the conceptual model in more detail. The conceptual model extends the Model-View-Controller (MVC) design pattern and serves as a base for Nested Visualization (visualization based on combining visualization methods where inserting elements of one visualization methods into elements of different one is possible).

The Model layer from the MVC pattern is represented by the entity Model, entity ModelElement and entity Property. Model is a container for Model Elements. Property is every pair of attribute and value. Through this entity the attributes to a Model or Model Element are assigned (e.g. colour of heading in the table).

Entities View and View Element can access these attributes and adapt GUI to values of these attributes. Entities View and View Element represent the View layer from the MVC pattern. The entity View is the GUI representation of given Model.

The Controller layer from MVC pattern is represented by the entities Recognizer and Consumer. Recognizer is every sensor which can recognize user's actions performed at a View or View Element to which this Recognizer is attached. Consumer is every service which can provide reaction to user's actions. Consumers usually change Model or Model Element.

The extension of the MVC pattern lays in entities Visualization, VisualizationElement, Connection and their relationships. Any visualization consists of the elements (e.g. the form contains labels and cells) and in the conceptual model reader can see the relationship between entities Visualization and Visualization Element with semantics: Visualization is composed of given Visualisation Elements. But even more important is the relationship between these two entities with semantics: Visualization is wrapped by given Visualization Element. This part of the model brings the ability of the AVS to visualize by Nested Visualization approach. Any visualization has to have its model and view. These facts are also mirrored in the conceptual model too. There can be created various connections between Visualization Elements (for example like connection between graph edge and its source and target nodes, between table row and its cells).

Visualization Methods Layer: Visualization methods layer is a set of components that implements various visualization methods.

Every implementation of any visualization method consist of components (*Visualization Method*, *Model*, *View*, *Visualization Method Element*, etc.) These have their counterparts in entities of the conceptual model in the language layer. By the implementation of the visualization method (adding a new visualization method) there also has to be implemented a simple interface. This interface provides functionality used for GUI building and controlling by the transformation layer. This functionality provides dynamics to the static structure which is expressed in the conceptual model of the language layer. All the functionality does not have to be described here but we can present an example: every implementation of any visualization method has its *Visualization Method* component and this must implement `addVisualizationMethodElement` and `removeVisualizationMethodElement` functions. These functions provide the ability to change the set of *Visualization Method Element*s in given *Visualization Method*. Thus the content of the output can be changed dynamically and can be adapted to actual demands.

Currently the visualization method layer of AVS consists of six visualization methods: Window, Form, Combo box, Spread sheet, Workspace and Dynamic mind maps. A sample of an output of the AVS is shown at figure 4. The AVS is here used in application MyNetScpe for the watching traffic in the network.

Transformation Layer: Transformation layer is quite universal thanks to the conception of the language and visualization methods layer. The structure of language layer does not change in time so the transformation layer is supplied by inputs with uniformly structured data. It uses Java Reflection API and from programmer's point of view can be seen as some kind of interpreter. It provides automatic generation of visualizations and GUIs from their descriptions.

On the figure 5 there is a schematically expressed that through elements (i.e. input) of specific language (where the elements are parts of the conceptual model of the extended MVC pattern) through simple automatic transformation a GUI with the Nested Visualization technique could be automatically generated.

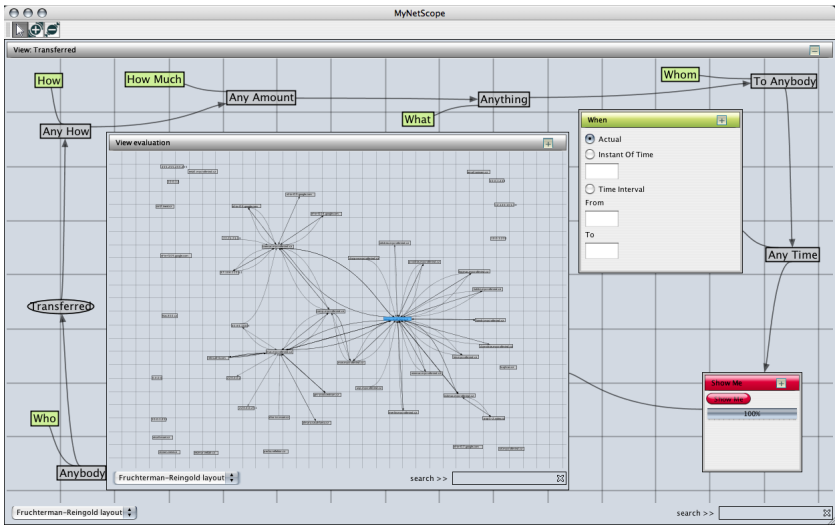

Fig. 4. Output of the AVS

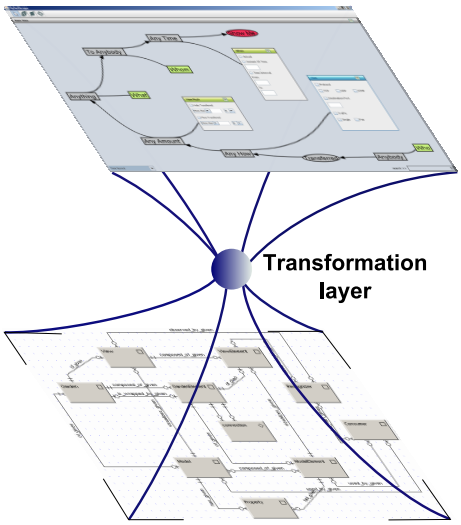

Fig. 5. Transformation layer

The main advantage of the transformation layer is the fact that an addition of new visualization method into the AVS does not require reprogramming of this layer. Only the extension of the visualization methods layer by implementation of the new visualization method is needed.

3.2 Extensibility in AVS

To minimize changes that are necessary when adding a new visualization method into AVS the following approaches were taken:

- AVS is divided into three layers – language, visualization methods and transformation layer
- all visualization methods are handled in a uniform way
- the Model-View-Controller pattern is used in the design
- the Model-View-Controller pattern is extended to allow to use the Nested Visualization approach
- AVS is language and platform independent

Let us have a closer look at remaining approaches as the first of them has been explained earlier in this paper.

Model-View-Controller: The basis for the AVS solution is the MVC design pattern. The model itself is representing the "raw" data to be displayed by given visualization method. Several different views can be linked to this model. These views observe the model and react by modifying themselves according to the observed changes. The controller is responsible for modifying the model according to user actions performed upon the view.

Model of the Basic Structure. Models and model elements in the AVS solution represent only the basic structure of the visualization. Thus they do not contain any data except their identifiers [13]. All data, attributes or properties are held in separate entities, in instances of subtypes of the Property entity (see Figure 3). The implementation allows to every visualization method to define its own property subtypes. This makes easier to handle various types of properties.

There are some general purpose properties that are common to the most of the visualization method elements (e.g. `textValue`, `isSelected`). However many visualization methods define their own properties, like `arrowShape` for an edge of a mind map.

The view or the view element in the AVS architecture plays the role of a wrapper to libraries or components which are implementing desired visualization methods.

We use the prefuse visualization toolkit [10] for graph-based visualizations for example. The view in our implementation of a graph visualization method wraps the prefuse visualization toolkit. Changes observed in particular properties of the model or model elements of the visualization are handed over to the wrapped prefuse component.

Adjustable View. A major criticism of the MVC design pattern aims at its inability to influence the view. There is no way how to set the properties of the view (e.g. colour, shape, etc.) by the model. One of the modifications of the MVC pattern that deals with this problem is the model-view-presenter (MVP) design pattern. It allows the presenter layer to modify just those properties of the view that the synchronization between model and view can not satisfy [9].

Our approach uses the concept of model or model element properties and their subtypes to tackle the problem of mentioning and changing the view. Every property of the visualization method that we wish to mention, i.e. allow the caller of our visualization service to modify it, has to be declared as a property (subtype) of the model or model element of the visualization method. An example could be the colour or width of an edge representing data flow in our network security monitoring application.

Thus we use the concept of model or model element properties to store data of the model as well as details about the visualization.

Controller – Recognizers and Consumers. We have extended the concept of the controller in several aspects. In classical MVC the controller is called as a reaction to events from the UI controls. The action to be performed is then hard-coded in the the controller code. The reaction to a UI event (mouse click, key hit) is hard-wired with firing the demanded application logic event. To change such behaviour means to rewrite the code of the controller.

AVS solution separates these categories of events. Recognizers implement listeners to low-level UI events – mouse clicks, keyboard shortcuts. They produce application logic events that are handed over to consumers registered to consume the kind of events produced by the recognizers they are registered with. Only consumers are allowed to change the properties of the underlying model or model elements. A recognizer can listen to several UI (low-level) event producers. A consumer can listen to several recognizers and one recognizer can supply several consumers with application logic (high-level) events. A consumer can modify several models or model elements properties.

This separation allows us to have a set of recognizers which can be rehang to different UI event sources without modifying a single line of code. Similarly consumers can be reconnected to alternate recognizers upon a command from the server. Supposed the required and necessary recognizers and consumers are implemented, we can change the behaviour of the user interface programmatically.

One of the situations that can be handled by this solution is the adjustment of the UI to user preferences for example. There can be several recognizers of an application-level action each one of them reacting to different UI actions. One of the recognizers listens to mouse clicks another one to keyboard shortcuts etc. So user preferences can be saved simply by storing only the connections between appropriate view (elements), recognizers and consumers. And by altering just these connections of the mention form preferences can be modified.

Uniformity: The possibility of uniform handling of visualizations and their individual elements makes AVS open to new visualization methods which will be required in future. Although it is easy to implement a new visualization method or its elements into AVS (similarly with removing). For each visualization a set of methods is implemented which is the same for every single visualization. The methods have the same interface across different visualizations. It makes the AVS modular and that is why services of AVS can be provided with different complexity depending on requirements of a user.

Language Independency: The transformation layer is written in Java. As it was mentioned in Section 3.2 the view or the view element plays the role of a wrapper. This wrapper can use the Java Native Interface to provide access to non-Java components. These wrappers encapsulate individual visualization methods or libraries. Thus using a concept of such wrappers makes the AVS language independent.

4 Conclusions and Future Work

Our work presents the Adaptable Visualization Service (AVS). The AVS is a part of a system that provides support in the field of network security. Requirements on such a system change very dynamically and particularly on its visualization service. First the type of information to be visualized is changing. Second also requirements on the visualization methods used to render given type of information can change in time. To minimize changes that are necessary when adding a new visualization method into AVS the following steps were taken: AVS is divided into three layers – language, visualization methods and transformation layer. All visualization methods are handled in a uniform way. The Model-View-Controller pattern is extended and used in the design. AVS is language and platform independent.

At the time of this writing, the first preliminary version of the complete system is being integrated and the whole system is still under development. The data used for system testing are acquired on Masaryk University network, connected to the Czech national educational network (CESNET). In our future work we will extend the set of provided visualization method implementations. Another direction of development is to extend the architecture by a renderer layer providing many visual features independently on particular visualization method implementation. These visual features will make the whole visualization more effective. The work is a part of a stream going to modelable and executable service systems with elements of artificial intelligence which is called Knowledge and Information Robots [20] based on principles of universal modelling and software construction [19].

References

1. Baxley, B.: Universal model of a user interface. In: DUX '03 Conference on Designing for user experiences, pp. 1–14. ACM, New York, NY, USA (2003)

2. Burkhard, R.A.: Towards a framework and a model for knowledge visualization: Synergies between information and knowledge visualization. Knowledge and Information Visualization 3426, 238–255 (2005)
3. Eppler, M.J., Burkhard, R.A.: A framework for the visual representation of knowledge. In: Multi-Conference Wirtschaftsinformatik, Passau, Germany (2006)
4. Hansen, S., Fossum, T.V.: Refactoring model-view-controller. J. Comput. Small Coll. 21(1), 120–129 (2005)
5. Lengler, R., Eppler, M.: Towards a periodic table of visualization methods for management. In: IASTED Conference on Graphics and Visualization in Engineering (GVE 2007), Clearwater, Florida, USA (2007)
6. MacDonald, M.: Pro WPF. Apress, (2007)
7. Molina P.J.: User interface generation with olivanova model execution system. In: IUI '04 9th international conference on Intelligent user interfaces, pp. 358–359. ACM, New York, NY, USA (2004)
8. Ruby on Rails, `http://www.rubyonrails.org/`
9. Fowler, M.: GUI Architectures, `http://martinfowler.com/eaaDev/uiArchs.html`
10. The Prefuse Visualization Toolkit, `http://www.prefuse.org`
11. JFCML - JFC/Swing XML Markup Language, `http://jfcml.sourceforge.net/`
12. Oškera, M.: Alternativní způsob reprezentace dat k objektovému přístupu. Master's thesis, Masarykova univerzita, Brno (2006)
13. Oškera, M.: Vztahově orientovaně úložiště. In: DATAKON 2006, Brno (2006)
14. Pettersson, M.: Designing the user interface on top of a conceptual model. In: CAiSe '95 7th International Conference on Advanced Information Systems Engineering, pp. 231–242. Springer-Verlag, London, UK (1995)
15. Quiroz, J.C., Louis, S.J., Dascalu, S.M.: Interactive evolution of xul user interfaces. In: GECCO '07 the 9th annual conference on Genetic and evolutionary computation, pp. 2151–2158. ACM, New York, NY, USA (2007)
16. Rehák, M., Pěchouček, M., Čeleda, P., Krmíček, V., Moninec, J., Dymáček, T., Medvigy, D.: High-performance agent system for intrusion detection in backbone networks. Cooperative Information Agents XI 4676, 134–148 (2007)
17. Santini, S.: Notes for the conceptual design of interfaces. Conceptual Modelling – ER 2006 4215, 413–423 (2006)
18. Phanouriou, C.: Uiml: a device-independent user interface markup language. PhD thesis, Virginia Polytechnic Institute and State University (2000)
19. Staníček, Z.: Univerzální modelování a konstrukce IS. PhD thesis, Masaryk University, Brno (2003)
20. Procházka, F.: Universal Information Robots a way to the effective utilisation of cyberspace. PhD thesis, Masaryk University, Brno (2006)
21. Terwilliger, J.F., Delcambre, L.M.L., Logan, J.: The user interface is the conceptual model. Conceptual Modelling – ER 2006 4215, 424–436 (2006)
22. Veit, M., Herrmann, S.: Model-view-controller and object teams: a perfect match of paradigms. In: AOSD '03 2nd international conference on Aspect-oriented software development, pp. 140–149. ACM, New York, NY, USA (2003)
23. Massó, J.P.M., Vanderdonckt, J., Simarro, F.M., López, P.G.: Towards virtualization of user interfaces based on UsiXML. In: Web3D '05 10th international conference on 3D Web technology, pp. 169–178. ACM, New York, NY, USA (2005)