Decomposition of Educational Presentation Systems

Georg Turban
Technische Universität Darmstadt
Germany
turban@informatik.tu-darmstadt.de

Max Mühlhäuser
Technische Universität Darmstadt
Germany
max@informatik.tu-darmstadt.de

Abstract

In this paper, we propose an extended version of our component-based framework for the development of educational presentation systems. The improvement is the application of a three-tier architecture. In contrast to related work, the introduction of new multimedia types, their rendering and storage can be completely decoupled from the core system.

1. Introduction

Educational presentation systems exceed the functionality of traditional presentation systems and typically support separation of views, ink-annotations and lecture recordings, for instance. The systems therefore deal with different multimedia types such as audios, videos, freehand drawings and images.

In this paper, we propose an extended version of our component-based framework (presented in [1]) for the development of educational presentation systems. The improvement is based on a three-tier architecture. Extensions that are built following the three tiers *presentation*, *business* and *data* can uniformly contribute to the core system. Such extensions can provide their specific rendering, handling and storage of multimedia content in order to support functionality and content that is not yet available within the core system.

2. Concept

The core presentation system is shown on the left side of figure 1. The *presentation* layer provides at least a presentation area. For a simple presentation system this area is shown to the lecturer and to the students. It consists of visualizations that are provided by extensions. The extensions render their content like layers in common 2d-graphical applications (more information about the rendering is provided in [2]) to this presentation area. The rendering is handled by the *EventDispatcher*. Commands such as *clear* and *store* force the rendering of all extensions. The *EventDispatcher* is also able to send commands to a dedicated layer. A *clear* for a single layer is typically applied to an annotation layer in order to wipe out all annotations while keeping the background, for instance. The *EventDispatcher* also receives events that are emitted from extensions. A layer that provides images of slides may emit the command *slide-transition* when applicable. The *EventDispatcher* then calls the layer *DataPool* in order to prepare the storage

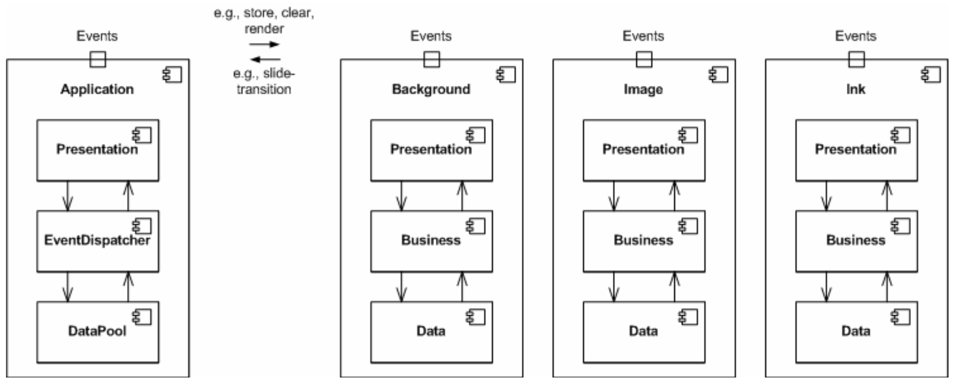
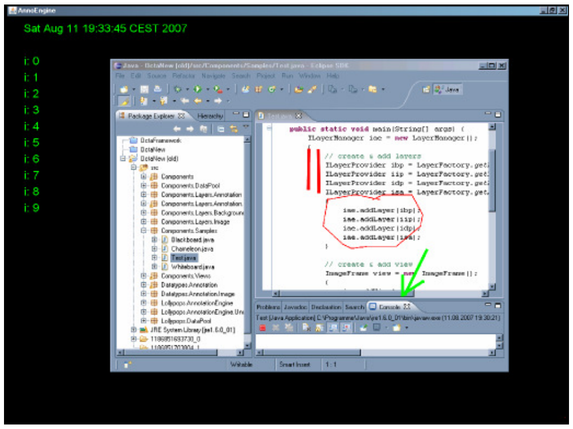


Figure 1: Core presentation system and extensions following a three-tier architecture.



```
Sat Aug 11 19:33:45 CEST 2007
i: 0
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
```

```java
public static void main(String[] args) {
    ILayerManager ise = new LayerManager();
    {
        // create & add layers
        ILayerProvider ibp = LayerFactory.get
        ILayerProvider iip = LayerFactory.get
        ILayerProvider idp = LayerFactory.get
        ILayerProvider isa = LayerFactory.get
        {
            ise.addLayer(ibp);
            ise.addLayer(iip);
            ise.addLayer(idp);
            ise.addLayer(isa);
        }

        // create & add view
        ImageFrame view = new ImageFrame();
        {
```

```
Test [Java Application] C:\Programme\Java\jre1.6.0_01\bin\javaw.exe (11.08.2007 19:30:21)
```

Figure 2: Snapshot of a sample set-up.

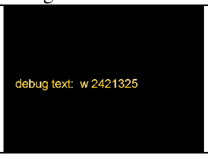
of information and content. In a simple scenario the *DataPool*-layer gathers the specific content from extensions by forcing them to store their content to an entry of the data pool.

3. Examples

The concept can be demonstrated using different sets of extensions. In addition, the extensions can be configured and some components can be exchanged on the fly. In figure 2, a snapshot of a set-up is shown that consists of following extensions: a background (always black), a remote framebuffer (shows the content of remote or local screens) and an ink (fast or smooth rendering) extension. In the given example, the remote screen contains the eclipse-IDE and arbitrary annotations. This set-up can be used as a basis for the development of distributed presentation systems.

Similar to screen-recordings, the rendered content can be continuously stored to disc, but augmented using meaningful navigational indices that can be obtained from slide-transitions. In contrast to screen-recorders our solution stores the content provided by extensions separately including more meaningful information (please refer to table 1 that contains data

Table 1: Sample entry of the DataPool.

ID	Content
0	background 0 0 0
1	 debug text: w 2421325
2	string br w 2421325
3	stroke 0 0 255 0 0 1.0 point 0 1186853828702 768 432 point 1 1186853828892 747 458 point 2 1186853828892 747 458 […] point 37 1186853833559 770 432 stroke 1 0 255 0 0 1.0 […]

from the first snapshot of figure 3) than available in flat images.

In figure 3, three other sample configurations are shown. The left one uses a simple ink-rendering component (points are highlighted and just connected by lines). The remaining two samples use an ink-rendering component for debugging purposes that displays the points and their corresponding timestamps. The center and right samples use the so-called *chameleon background* component (before and after switching to a whiteboard-surface) that allows switching between different colored surfaces.

4. References

[1] G. Turban, and M. Mühlhäuser, "An Open Architecture for Face-to-Face Learning and Its Benefits", in: Proceedings of the 8th IEEE International Symposium on Multimedia, San Diego, CA, USA, 2006.

[2] G. Turban, and M. Mühlhäuser, "A Framework for the Development of Educational Presentation Systems and its Application", in: Proceedings of the 1st Workshop on Educational Multimedia and Multimedia Education, in conjunction with ACM Multimedia, Augsburg, Bavaria, Germany, 2007.

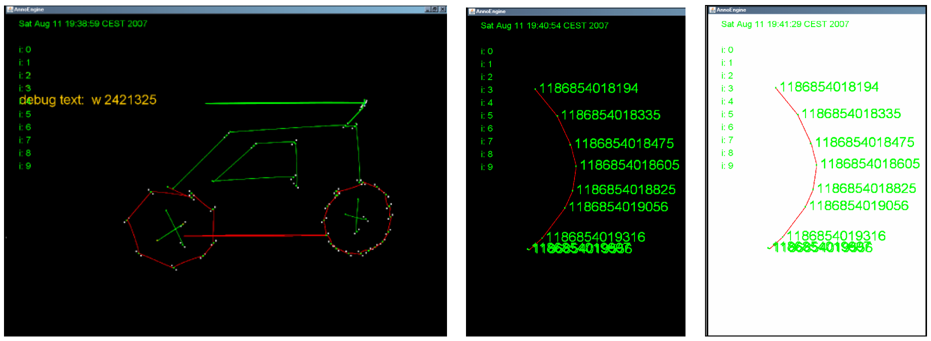


Figure 3: Different samples of the presentation system and underlying components.