HCI-Patterns for Developing Mobile Apps and Digital Video-Assist-Technology for the Film Set

Christian Märtin[1], Anthony Stein[2,3], Bernhard Prell[2], and Andreas Kesper[2]

[1] Augsburg University of Applied Sciences, Faculty of Computer Science,
An der Hochschule 1, 86161 Augsburg, Germany
Christian.Maertin@hs-augsburg.de
[2] Vantage Film GmbH, Digital Division,
Fuggerstraße 7, 86150 Augsburg, Germany
{AnthonyStein,BernhardPrell,AndreasKesper}@vantagefilm.com
[3] University of Augsburg
Department of Computer Science
86135 Augsburg, Germany

Abstract. Digital cinema technology is now widely accepted by directors, directors of photography, producers, film crews, and during the post-production process. On the film set high-resolution digital motion picture cameras have entered the field. In order to exploit the full creative and organizational potential of the advanced digital production technology and to support the whole shooting process, digital video-assist systems are connected to the cameras, monitors, and auxiliary components on the set to form a computer-supported film set (CSFS). The CSFS around Vantage Film's PSU® family of advanced video-assist systems offers intelligent support for all the roles and tasks on the film set. This paper focuses on the design of the PSU® product generations. Contextual design, agility, and patterns, both for designing control and user interface functionality, have been used extensively in the development process. This is demonstrated for the iPad-based mobile PSU® Satellite and some GUI patterns that were used for different features of the touch-screen based user interface.

Keywords: Digital video-assist, design patterns, HCI-patterns, digital motion picture cameras, iPad, iOS, touch-screen user interface, computer-supported film set.

1 Introduction

For more than a decade digital video-assist systems have been used to support and optimize the production process of feature films and commercials directly on the film set. Such highly interactive systems are responsible for take recording and replay, rendering and pre-screen simulation of special effects, rehearsal purposes, and for improving the user experience of the entire film crew [9], [3].

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2014, LNCS 8510, pp. 320–330, 2014.

A system that can integrate most of the digital hardware, software functionality, and user-assist technology for all the roles on the set within a common distributed architecture, is called a computer-supported film set (CSFS). When we developed our first CSFS, the PSU®-1 video assist system, using a contextual design process [4], [1], we came up with some solutions for the resulting high-end video-assist-system, like touchscreen-based user interface or content-based retrieval of film takes that even today still define the state-of-the-art in video-assist technology.

Our imagination and the feedback from early contextual inquiry showed us the direction for our future work and provided us with ideas, which roles were needed in the shooting process and which film and IT components should be integrated into a CSFS. At that time we already envisioned and prototyped the *Director´s Pad,* a tablet based video-assist client. Only today, however, embedded and wireless technology provides the necessary performance, energy efficiency, reliability and software flexibility for building high-performance mobile video-assist clients for professional use in all imaginable contexts.

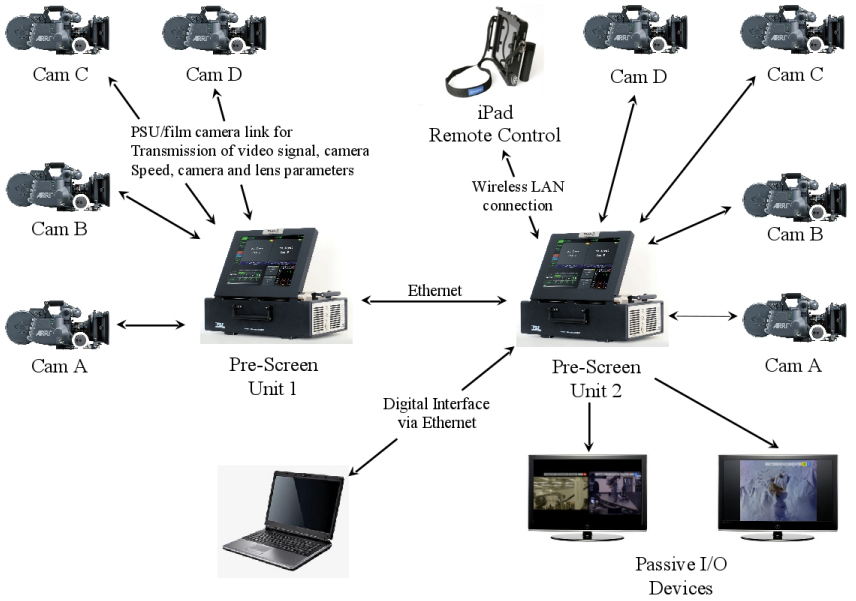

Fig. 1. Computer-supported film set based on the PSU®-3X HD digital video-assist system

The current PSU®3-X HD based CSFS, as shown in figure 1, combines client-supported video recording features with simultaneous multiple camera support, intelligent software functionality for on-the-set effects simulation and rendering (slow-motion, time-lapse, ramping, filtering, mix-and-overlay of takes, blue-, green-screen effects, editing, 3D-support, etc.), shooting day organization, rehearsal mode, and hard- and software-support for active camera control and communication [2].

In the following it will be shown that our development process for advanced video-assist software, could profit enormously from the use of different pattern categories.

We will outline, how contextual design, object-oriented modeling and agile development practices helped us to attain the high quality standards, necessary for a whole class of successful real world products.

2 Mixing Contextual Design and Agile Development

When we started to develop software and digital technology for supporting the movie production process in 1999, we first had to get acquainted with the environmental, contextual and functional requirements of the people working on the film set, their roles, their tasks, their goals, their wishes, and their communication behavior. We formed a joint team of computer scientists and film professionals and started the project that should lead to a widely accepted and reliable CSFS solution. Already in the early planning phase, we de decided to set up a contextual design process [1] – at the time a relatively novel development method for user-centered systems.

2.1 Design Process

After the first interviews and contextual inquiries with directors and DoPs we did some paper mockups and experiments with desktop computers and various interaction techniques. We defined the major use-cases and specified the basic video-assist-tasks. We immediately abandoned mouse and keyboard interaction, as we learned from our interviews, that these devices were not accepted by the crews on the film set. We relatively straightforwardly decided to base the user interface completely on a touch-screen solution. We then built an early touch-screen based prototype with limited performance and only the visualization features and interactive properties necessary for demonstrating the key functionality, e.g. how to record takes, or how to replay takes in slow motion.

About one year after project launch the prototype was presented to a large audience of directors of photography (DoPs), camera operators, directors, and film crew members, at an international industry trade-show for film equipment. The prototype got enormous positive feedback and the final decision for product development was made. Within two more years a small team of never more than ten developers (including software and hardware development, user-interface design, and mechanical construction) with accompanying input from film professionals at each critical stage created a pre-product prototype of the final system that was tested under real-world production conditions by directors and DoPs around the world [4].

Most usability problems resulting from a lack of communication between software engineers and the final users, i.e. our film experts, could be avoided by having the film people evaluate each new interactive feature on the pre-product prototype immediately after the feature was available. This method worked for the first product version of the system, the PSU®-1, and was used in all of the following video-assist products since then. Software features that will be integrated in future product versions today are first integrated into software updates for the current product version and tested and evaluated on real sets. The other way round, ideas for

innovative features often come from the people on the set. As we listen to them, our agile process allows us to implement new features rapidly and introduce them in one of the next software updates that are distributed by a remote maintenance process.

It soon became clear that with a rapidly growing interactive functionality a layered design approach was necessary for the user interface in order to maintain the overall look-and-feel of the system. Although the visual design undergoes evolutionary changes from product generation to product generation, each new product will still present its basic take recording and replay functionality on the main screen after power-up. More complex features are hidden below the surface. The more specific or sophisticated the functionality becomes, the more effort is needed by the operator to arrive at the desired level of functional detail. However, several design and HCI patterns guide our layered structuring of the software and the design of the end-user navigation. Examples of such patterns are given in sections 3 and 4.

2.2 Technology Context

In the 15 years since we started to develop systems and components for the computer-supported film set the greatest technology revolution was the migration of the film industry from celluloid film to digital film recording. Today, most advanced video-assist systems are used in combination with highly sophisticated digital motion picture cameras, like the ARRI Alexa [12], the RED Epic [13], or the Sony F65 [14] to name only a few. In contrast to earlier professional electronic cameras, these digital film cameras already are equipped with integrated support for HD video-out signals that can be sent to the video-assist system over the HD-SDI interface. Earlier cameras only provided analog SD video-signals for video-assist purposes. The high-definition 2K or 4K high contrast and high luminosity raw signals of today's most advanced film cameras are used in post-production for preparing the final movie and are not directly used on the set. They have to be stored in separate high-capacity film storage servers.

However, the video signals that arrive over the HD-SDI interface are of remarkably high quality, i.e. available as HD video signals. They can directly be exploited by the digital video-assist technology on the set. The current product generation, the PSU®3-X HD, is able to record up to four takes simultaneously and at the same time replay up to two recorded takes (on different external HD screens). The high computing performance that is needed by the CSFS is made possible by an embedded Intel Core-i7 quad core processor, Nvidia GPUs, several custom-designed FPGA-boards for frame grabbing and visualizing takes, the Linux OS and by highly optimized software using C++, Qt, and OpenGL. The power envelope of the system is nevertheless relatively low (100W). Therefore, the system can be battery-powered for several hours.

To further improve the flexibility on the film set and allow directors, DoPs and crew members the direct interaction with the PSU® systems while moving around at the location, a mobile extension to the CSFS, the iPad-operated PSU® Satellite was recently announced. Its software design is discussed in the following chapter.

3 The Usage of Design Patterns within the iOS Domain

The application discussed in this chapter was developed using a model-based approach, where design patterns serve as the models. On the basis of the so-called *PSU® Take Manager,* which is one of two currently developed iOS applications with the aim to increase the mobility of the CSFS, some well-known design patterns are going to be described for the context of iOS programming. On the one hand the *model view controller (MVC) pattern* – also sometimes denoted as an architecture pattern – gets illustrated. On the other hand some other useful design patterns become the subject matter. To reach the goal of increased mobility on the CSFS, the iOS Device (here the PSU® Satellite, based on the iPad) interacts with the Digital Video-Assist System PSU®-3X HD. To provide an understanding of how this challenge is faced, a short overview of the rudimentary feature set is given [7], [8].

As mentioned above, two independent applications are under development. The *PSU® Take Manager* (fig. 2, left) helps directors to replay already shot takes stored on the PSU® main device. Such takes can also be stored and organized in the iPad's internal storage. To find the right scene to replay, a filter mechanism for a more comfortable browsing through all available takes was integrated. During replay takes can also be viewed frame-by-frame.

Another application, called *PSU Streaming* (fig. 1, right), offers the ability to look at up to four independent live-streams from cameras connected to the PSU® or disc playback. Furthermore some bi-directional conversation functionality was implemented, so that the director can view the scenery from different viewpoints and give instructions to the (human) video-assist operator "over-the-air".

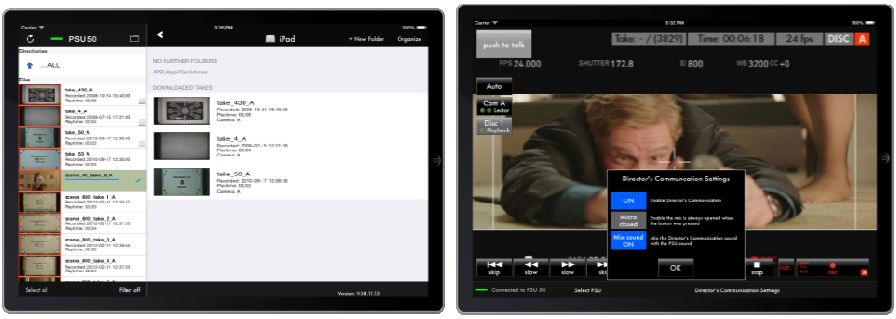

Fig. 2. PSU Take Manager screenshot (left) and PSU Streaming screenshot (right)

For the purpose of iOS-App programming a framework provided by Apple called *Cocoa Touch* comes into operation. This framework makes great use of software design patterns from object-oriented design (OOD) and was an ideal platform for implementing CSFS-mobility. Some of the patterns are described in the following. Figure 3 illustrates the basic concept of the MVC pattern within the iOS domain.

The MVC pattern resembles a three-tier architecture with a database-, a business logic-, as well as a presentation-layer. The model corresponds to the database layer

whereas it is not restricted to database connections at all. It could provide data from the web via HTTP or, e.g., from a persistent file. In the case of the *PSU® Take Manager* the model is represented by a WebDAV client, which also handles the connection to the PSU®-3X HD. Over an established WebDAV connection meta-information about the takes stored on PSU can be requested. Files can be accessed via a GET-request. The returned meta-information is stored in the iPad's memory and prepared for further usage within the application.

The middle layer – the controller object – handles the delivery of requested data to the view layer on top. It collects the requested data from the model (pulling), transforms it into an adequate representation and returns the well-formatted data via the return parameter of the callback method invoked by the view.

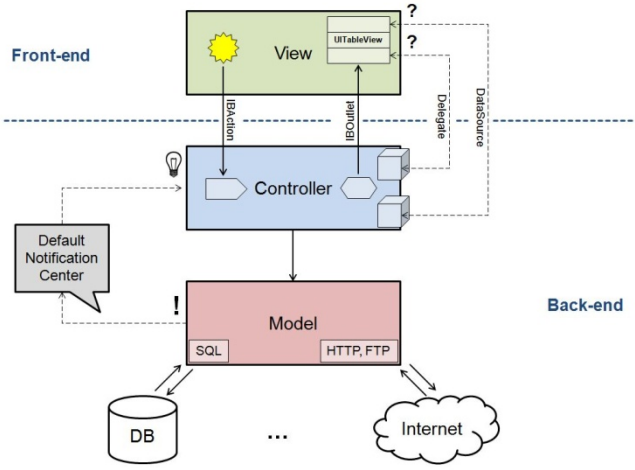

Fig. 3. An MVC illustration for the Cocoa Touch Framework [8]

Figure 4 shows a concrete model-based implementation of the MVC pattern on top of the task of an initial WebDAV request when the PSU® Take Manager App gets launched. The semantics of the red dyed actions is that these actions are invoked or provided by the iOS Framework. The UML signal element was used to illustrate notifications (described below) as well as events on which callback methods will react.

Using callback methods, invoked when needed by the system, is one approach for updating data presented to the user. Another method is pushing new information to the view directly. Therefore Apple provides a macro called *IBOutlet* that allows the graphical interface builder to bind an interaction object, e.g. a label, to a corresponding instance in the source code.

Let's consider the case of a progress view. While downloading a take, the PSU® delivers small data packages that are collected and stored by the model. The view is presenting a so-called *UIProgressView* to let the user know about the status of the download progress. Each time a new data package arrives, the progress has to be

calculated again and passed to the controller that manipulates the view in a direct manner by accessing the bound IBOutlet member.

Pushing the information from the model to the controller is possible through a mechanism called *notification posting*. The sender (here the model) posts an *NSNotification* object with appropriate information to a central message receiving station object (*NSNotificationCenter*). This central instance routes the notification to all objects that have been registered as observers for such a kind of notification. This is an adapted implementation of the well-known *Observer Pattern* [10], [11].

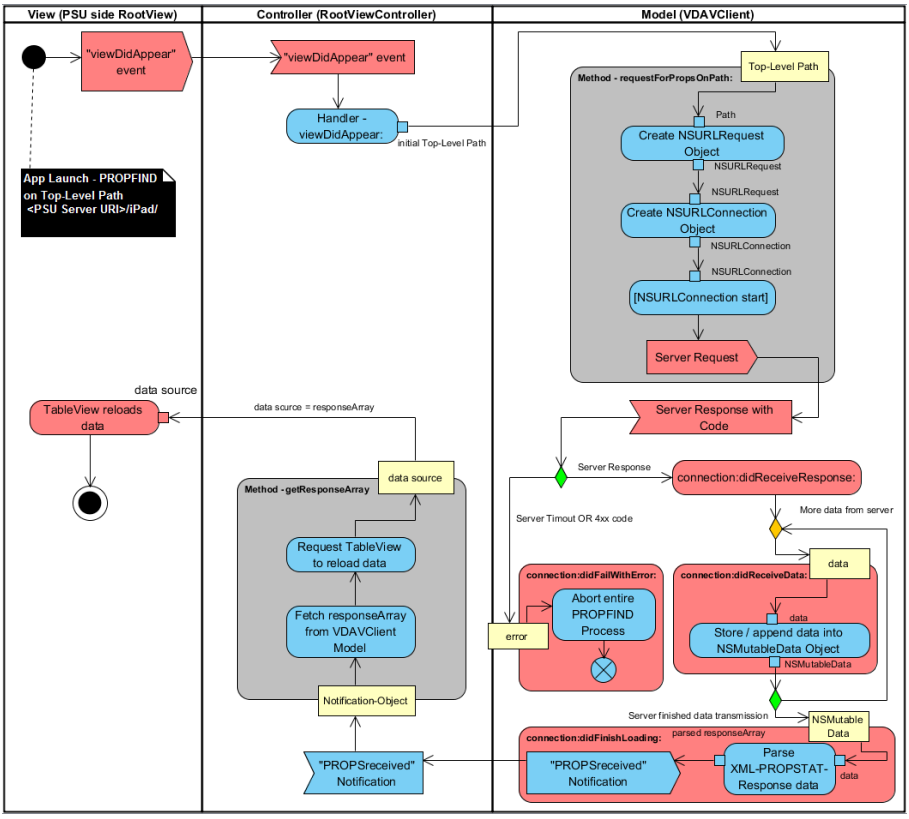

Fig. 4. Activity Diagram – MVC Implementation of WebDAV Request Mechanism [8]

The Cocoa Touch framework makes use of the *Delegation Pattern*, too. The controller responds on requests sent by the view or reacts on notifications about changing data from the model. The implementation of the already mentioned callback methods invoked by the view is mandatory, because these methods are defined in a *protocol*. A protocol equals the concepts of interfaces or pure virtual methods and is a language-level feature of Objective-C. Protocols make it possible to let two objects with different public interfaces communicate with each other. Thus, protocols can be interpreted as an instance of the *Adapter Pattern* [10], [11]. A delegation, e.g.,

happens when a UI control element is asked how to appear. A delegate object could be set up conforming to a special purpose delegate protocol. An example is the *UITableView* object. A table view needs to know how many rows and sections it will have, how the cells should appear (height, color, etc.), whether there should be a section header or footer describing the individual sections or not, and so on.

The Cocoa (Touch) framework also considers a core problem of graphical user interfaces. A developer of a GUI puts so called interaction control elements on the screen so that a user can interact with the software in the background. The programmer in fact doesn't know at which time or how often a user decides to use the element just as a developer of a GUI library can't know how, e.g., the usage of a button is intended by the developer of a GUI. The flexibility to face this issue is one of the purposes the *Command Pattern* was introduced. When using this pattern a method invocation gets encapsulated by an object. This object usually holds the receiver (also denoted as *target*) of the message, as well as the identifier (*selector*) of the method which should be invoked. The GUI developer defines, which message to which receiver shall be executed and every time a user interacts with a control element, an invocation object is generated dynamically. That, in turn, makes it possible to trigger a method invocation at runtime. Apple conforms to this pattern with the *Target-Action-Mechanism* as well as through *NSInvocation* objects [11].

Another design pattern introduced by Erich Gamma et al. (aka The Gang of Four, see also [10]) is the *Composite Pattern*. This approach allows the programmer to use and implement whole/part-hierarchies of objects. That means that an abstract composite class specifies behavior of individual objects and compositions (container) of such individual objects at the same time. Thus the individuals and compositions of individuals could be used uniformly. With this technique it is possible to build tree-structured hierarchies of objects, whereby the developer doesn't have to handle of which type an instance of a tree-branch (composition) or tree-leave (individual) is.

A more specific example is the view-hierarchy of a graphical user interface. The *UIView* class is a composite class, because it is the base class of most interaction elements like sliders, buttons, etc. If a superview containing some subviews, which are in turn superviews of further subviews (tree-structure), is asked to redraw itself, the redraw message is propagated to all subviews until the leaves are reached. Thus the internal treatment of branches (container) and leaves (individual subviews) is the same [11].

To keep the limited scope of this paper, the focus was set on such design patterns that are related to the User Interface most closely and thus meet the topic of human computer interaction. Of course, some more instances or adaptions of well-known design patterns exist within the Cocoa (Touch) framework. For more details on the mentioned patterns and the usage of them within the iOS-Framework we refer to [10] as well as [11].

4 User Interface Patterns for the Digital Video-Assist

Another area for the successful application of patterns in the CSFS development process is the touch-screen user interface of the PSU® main device. During the evolutionary development process of the PSU® family of video-assist-systems the user interface had often to be adapted to new functionality and new interaction styles. Figure 5 shows an example of the slider user interface pattern.

The slider pattern is always used when a parameter has to be intuitively set, without the necessity for entering an exact value. Implemented as a take slider, it is used to navigate to a specific position within a take. With the slow-buttons, each frame can be directly located. Another implementation would be the use of the slider pattern for the intuitive setting of the overlay intensity, when two image-layers are superimposed as, e.g., for 3D cinema simulation). Finally, the slider pattern can be used for the intuitive setting of brightness, contrast and gamma parameters.

Fig. 5. Implementation of the slider pattern as a take-navigation-slider

Another HCI pattern example that is reused for different functions ist the *+/- display element*. It serves to enter a new value that is not too far away from the current value. If pressing the buttons longer, the rate of the value changes is accelerated. The graphical representation of the pattern can also be seen in figure 5, where it can be used to set the frame rate (original camera speed, or some other value for simulating slow motion or time lapse).

The same pattern is used for setting the length of short takes, that are ignored by the system and will be deleted automatically, or for the fine-tuning of the cooling fan in order to avoid too much noise when shooting takes, or the setting of the microphone volume. If the user presses on the display, a numeric keypad automatically appears for entering an exact value.

Many other user interface patterns were applied for getting a consistent look-and-feel and for reducing the user-interface complexity for the end user [6].

5 Conclusion

Design patterns and HCI patterns can contribute heavily to the successful construction of complex interactive systems. This was shown for the development of an iPad-based mobile video-assist client. The applied structured patterns available in Apple's Cocoa Touch library served as design models and we assume they facilitated the development of several communication tasks between the PSU® main system and the client. At the same time, the use of patterns could accelerate the development process and ensure good coding quality. It was also demonstrated by examples that HCI patterns can be used for keeping navigation and interaction consistent from product generation to product generation and lead to high acceptance by end-users. Our experience with the systems has shown that users in productive environments appreciate, if they quickly recognize functions and can use new functionality intuitively, because interactive behavior is controlled by usage patterns, they already know. Patterns therefore also serve as a means for reducing complexity in rich interactive environments.

References

1. Beyer, H., Holtzblatt, K.: Contextual Design. Interactions, 32–42 (January + February, 1999)
2. Fauer, J.: Vantage PSU-3X HD Digital Video Assist. Film and Digital Times (55-56), 76–77 (2013)
3. QTake (2013), `http://qtakehd.com` (retrieved on September 13, 2013.)
4. Märtin, C., Prell, B.: Contextual Design of a Computer-Supported Film Set: A Case Study. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 392–405. Springer, Heidelberg (2003)
5. Märtin, C., Prell, B., Kesper, A.: A New Generation Digital Video Assist System with Intelligent Multi-Camera Control and Support for Creative Work on the Film Set. In: Tavangarian, D., Kirste, T., Timmermann, D., Lucke, U., Versick, D. (eds.) IMC 2009. CCIS, vol. 53, pp. 331–332. Springer, Heidelberg (2009)
6. Märtin, C., Prell, B., Schwarz, A.: Managing User Interface Complexity and Usability in Computer-Supported Film Sets. In: Proc. of HCI International, Las Vegas, Nevada, USA, July 22-27, vol. 3. Lawrence Erlbaum Associates (2005) Human-Computer Interfaces: Concepts, New Ideas, Better Usability, and Applications Series
7. Märtin, C., Stein, A., Prell, B., Kesper, A.: Mobile App-Support for Advanced Digital Video-Assist Systems in Computer-Supported Film Sets. In: Proc. 6. Forum Medientechnik, Fachhochschule, St. Pölten, Austria (2013)

8. Stein, A.: Evolutionäre Entwicklung einer proprietären iPad-Applikation für die mobilitätssteigernde Interaktion mit dem digitalen Video-Assist-System PSU®-3, Bachelor Thesis, Augsburg University of Applied Sciences (2012)
9. Vantage Film GmbH (2013), http://www.vantagefilm.com (last access February 4, 2014)
10. Gamma, E., et al.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
11. Apple Inc.: Cocoa Fundamentals Guide, pp. 165–208 (September 2013)
12. http://www.arri.com/camera/digital_cameras/ (last access: February 4, 2014)
13. http://www.red.com/products/epic (last access: February 4, 2014)
14. http://pro.sony.com (last access: February 4, 2014)