A Model-Driven Interactive System

Hao Wu and Qing-yi Hua

School of Information Science and Technology Northwest University,
710069 Xi'an, China
{wh,huaqy}@nwu.edu.cn

Abstract. Model-driven interactive system development mainly solves the user interface development problems of the diverse and heterogeneous client devices in the new computing model such as pervasive computing and cloud computing. This paper analyzed the problems that existed in this development approach, and put forward an architecture pattern ALV+. This pattern was integrated with CAMELOAN reference framework. So a novel model-driven interactive system development approach was designed. This approach divided the user interface into several models: abstract interface, concrete interface, device interface and event description, which made the presentation and transformation of the models simple, and decreased the redundancy among the models. Then a case study was used to verify the effectiveness of the approach, and confirmed that this method could solve the problems of the model-driven approach to some extent.

Keywords: new computing model, model-driven, architecture pattern, CAMELOAN reference framework.

1 Introduction

Model-driven interactive system development is considered to come to the third stage of its evolution process [1]. In this stage it mainly solves the user interface development problems of the diverse and heterogeneous clients in the new computing model such as pervasive computing and cloud computing. These development methods have obtained some achievements, but there are some problems in these methods: 1) A weakness of model-driven development approaches is that it seldom uses the technology of the mature architecture pattern, so it is difficult to apply the research results of architecture pattern in recent 30 years, so as to bring so many problems about the design, development, test, evaluation, deployment and evolvement of systems [2]. 2) In the model-driven development approaches, the presentation and transformation of models are complex and difficult to master. These approaches produce the limited user interface that the development tools are supported [3]. 3) In quite a few model-driven system development methods, the models are divided into interactive models, presentation models, platform models and so on. Although the complex sorts can definitely describe the properties of use interface from different views, there is redundancy that is difficult to erase among many models. The redundancy leads to the cumbersome use interface generation process. 4) Model-driven development approach is a step wise refinement approach to address the static appearance of user interface, but there are weaknesses in solving the dynamic behavior and control problems, because the response and disposal of events and the data transformation between user interface and the application computation are often closely interrelated with final interface, resolving these problems at the abstract level often leads to interweave the user interface and the application computation.

This paper puts forward a model-driven development approach for interactive system, which supports the auto-production of user interface for multi-devices, and aims to do some exploratory research for the aforementioned problems.

2 Related Works

2.1 Interactive System Architecture Pattern ALV

Interactive system architecture pattern ALV shows as Figure 1 [4]. In this pattern, shared abstract denotes the application computation of system, view denotes the user interface of system, and link links the abstract with view by the constrains-based ways. ALV pattern has a good effect in supporting CSCW. The front devices in CSCW environment resemble client devices in new computation mode: there is a share application service in back end. But in ALV-based CSCW environment, front devices are fat client that have the strong interface presentation and computation power. They have the same interactive modalities; all are the standard configuration of PC. Whereas in the new computation mode, the presentation and computation power of front devices have the remarkable differences. They may be fat client, but they may be the thin clients that have the weak presentation and computation power and have the different modalities. So ALV pattern couldn't apply to the new computation mode directly, it needs some expansions and improvements.

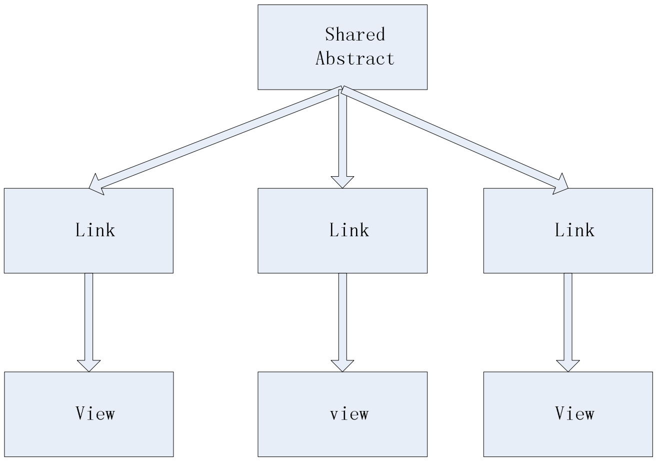


Fig. 1. This Figure shows architecture pattern ALV

2.2 CAMELEON Reference Framework

CAMELEON reference framework [5] shows as figure 2. In this framework, the development of interactive system begins with building concept and task models. Then the framework produces the abstract models from these models. The abstract models transforms into the concrete models by the models mapping. In the last, the final user interface is produced by the concrete models. CAMELEON reference framework doesn't give the specific models; it just refers to the four stages of the life periods of the user interface development. Many a model-driven user interface development works were based this framework [1] [6]. This framework is a top-down development mode for the new system development. The problems that this framework exists have been explained in the preceding part of the text. This paper presents a method that doesn't preclude this approach, but it only limits to the building of static user interface. For the dynamic parts of user interface, it adopts the down-top design ways, it makes a clear separation between user interface and application computation.

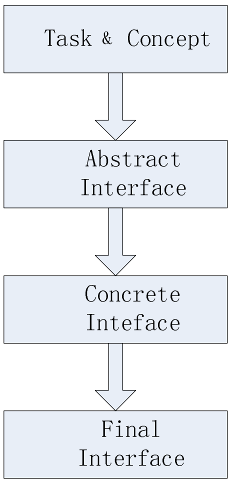


Fig. 2. This Figure shows the CAMELOAN reference framework

3 Architecture Pattern ALV+

For using ALV pattern in the new computation mode, it has to be extended and evolved. This paper names the pattern extended as ALV+ to illuminate it was extended by the ALV, showing as figure 3. Shared abstract still denotes the application computation, *view* denotes the user interface of the client also. The difference with ALV mainly lies in that the function of the *link* doesn't link the abstract and the view by the constrains-based manner no more, and the transverse lines between the links is added to support the dynamic switch user interface between the devices.

In the new computation mode, the client devices that have the different computation, presentation power and interactive modalities access the same service.

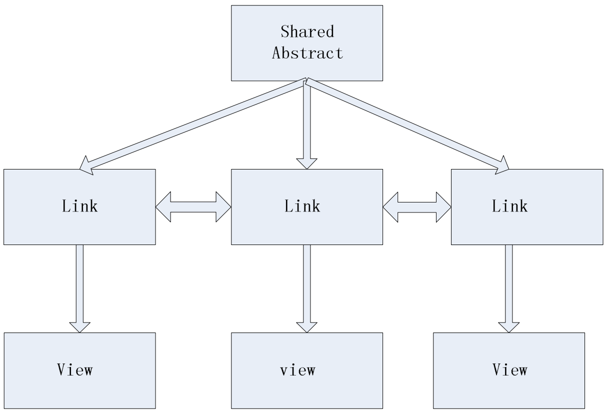


Fig. 3. This figure shows the ALV+ pattern

The problems that ALV+ solves are that the same application service needs to develop the different user interface for these devices that have the large differences, and the user interface may be switched between the devices. So, the function of the link unit is modified as having the function of "the application service transformer". Like that the transformer transforms the high voltage electrical current that is carried from power plant into different low voltage electrical current according to the devices' needs, the application transformer (link) reifies the abstract interface that is passed by application service with the concrete interface that exists link unit for the different client devices, so the final interfaces for these devices are built. At the same time, while the user interface of one device need to switch to the user interface of another device at running time, it saves the states data of the former, and transmits to the latter to dynamically produce corresponding user interface(show as the transverse line of figure 3). While deploying the components of ALV+, the abstract, link and view can locate at the different devices like ALV, they link together by the network. Also according to reality needs, the abstract and link, or link and view could be located at the same machine.

4 Model-Driven User Interface Development Approach

Our model-driven user interface development approach shows as figure 4. In this figure, application service sends the abstract interface to the link unit by needs. Abstract interface is the logic description of the final interface (a window or a HTML page) that is independent of platform and program language. After the link unit receives this abstract interface, it finds out the concrete interface that reifies this abstract interface in link unit. The concrete interface is dependent on the platform and program language. In the C/S architecture, after abstract interface combines with concrete interface, device interface is produced. Device interface is the accurate description of the final interface. Device interface is distributed to view unit. It could make network transform easy. View unit automatically generates final interface by the device interface. In the B/s architecture, after abstract interface combines with concrete interface, the HTML page is automatically generated. The events that the final interfaces give birth to are defined by the event description. Event description.

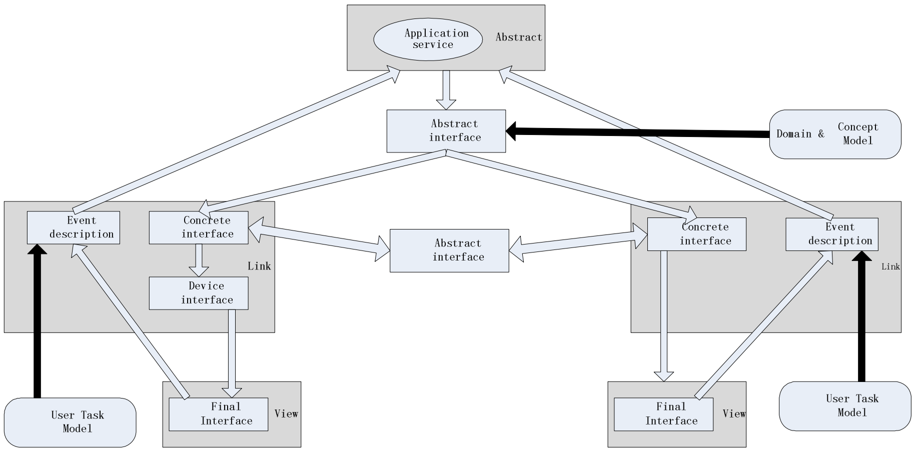


Fig. 4. This figure shows model-driven user interface development approach

Application service. After these data are handled by application service, they may return a new abstract interface to the link unit. While the interface needs to switch between the devices, the link unit of the source device collects state data and generates the abstract interface. The abstract interface is sent to the link unit of the target device. The target device then transforms the abstract interface into the final interface in the target device.

This approach uses the ALV+ architecture pattern: the application service in the top of the figure is the shared abstract of the ALV+. The concrete interface, device interface and event description in the middle of the figure is the link of the ALV+. The final interface in the bottom of the figure is the view of the ALV+. The abstract interface between the two concrete interfaces is the carrier of the data that the link levels of the two devices need to transform while the interface of the one device needs to switch to the other device at run time. In this approach ALV+ works as the organization of the system components and the guidance of the components deployment. This is the embodiment that architecture pattern in the model-driven interactive system development exert influence.

This approach also uses the CAMELOAN reference framework: the approach includes the task, conception model, abstract model, concrete model, and final interface. The differences with the conventional CAMELOAN reference framework use are: 1) In this approach two components, device interface and event description, are introduced by the needs. 2) This method combines the top-down development way with down-top development way. Task and conception model leads to abstract interface. The foundation that the abstract interface is produced is the needs of the interaction between application service and final interface [7-9]. This is a top-down ways. The task model guides the design of the event description. The event description could clearly define its contents after the concrete interface confirms the component type that produces the event. This is a down-top ways. In this process, although the final interface doesn't clearly present, it should be assumed in the design's brain while designing the concrete interface. 3) CAMELAON reference framework gives a development process [10], this approach adopts its basic conception. But our approach is a flow. This flow mainly uses the undertone arrows to link together in the figure. The fuscous arrows show the leading relation.

This approach has these advantages: 1) because the most application computation works have finished by the application service, and the final interface just has the function of data transform that the application computation needs, the requirement for computation power of devices is furthest lowered, the difference among client devices computation power in multi-device environment is shielded [11]. 2) Because introduction of interactive requirement leads to the abstract interface model, every abstract interface corresponds to a frame/page interface, which makes the presentation of the interface straightforward. At the same time, the concrete interface straight depicts the abstract interface, there needn't complex model transform arithmetic. The express of the abstract interface and the concrete interface doesn't have redundancy. 3) The abstract interface is independent of the platform and the program language, in the event description of the different devices the data format is uniform, so the application service is independent of the specific platforms and languages. The works that the system supports new devices are simple, the only need is to add the new link unit to the system [12]. 4) The programmers that have different duty write the different parts of the interface description, which makes the work assignment clear. For example, application programmers write abstract interface, interface designers write concrete interface. 5) The building of concrete interface could aim at the different presentation power and modality of devices, so the requirements of multi-device environment are meet.

The description of the abstract interface, the concrete interface, the event description and the device interface uses XML language to express.

4.1 Abstract Interface

Abstract interface is the logic level of the interface, which has the independency on platforms and languages. It gives out the type of interaction (ToI), the content of the interaction, and the correlative attributes of interaction, and so on. By analyzing the existing services and correlative papers, we define eight kinds of ToI: input, output, select, modify, create, destroy, start and end. In the work of this paper, we assume that these ToIs could cover all the possible interaction between application services and user interface.

Abstract interface is usually composed by the forest which makes of groups and ToIs. The child nodes of the group may be groups or ToIs, but ToI doesn't have child nodes no more. Group shows the construction and organization relations between ToIs. A ToI denotes a interaction behavior that could happen between application service and user interface, which presents one or more interface components while being implemented.

4.2 Concrete Interface

Concrete interface is the physical level of interface, which mainly presents the static appearance that is special to some type of device interface. It is closely correlative to the interface language (Such as Java, html).

Concrete interface describes the ToIs and groups of the abstract interface. In concrete interface, the description for the elements of abstract interface divides into three levels: group, type and name. The description for the group would work on all the nodes that belongs the tree or sub-tree whose root node is the group; The description for the type makes the ToI that is the type have same characters; The description for the name aims to the ToI that has the given name.

Concrete interface could be sorted as basal and specific interface. Basal concrete interface includes the default description for every type of ToI, which could be reused by several services. Specific concrete interface is designed for the specific demands of one or more service, which has high priority in the interface combination. If a ToI or a group of a abstract interface doesn't have name mapping, then basal concrete interface is used to achieve the specification for them.

From the definition of the abstract interface and the concrete interface and their interrelation we can see that abstract interface and concrete interface don't have complex model presentation, mapping and transformation arithmetic, but the description ability doesn't be lowered.

4.3 Device Interface

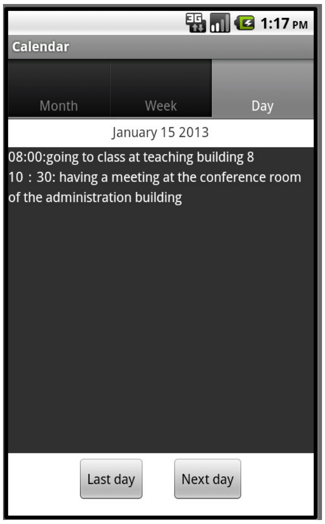
Device interface only appears in the application of C/S architecture, which is the whole description of final interface, and combined by the abstract interface and concrete interface. But the combination isn't the simple combination. We throw off the redundant components of the interface description in the combination process. Device interface is automatic produced by the generation engine. The generation engine checks the solecism and semantic error of the description languages. Device connection unit would transfer the device interface to the client, which is more simple and convenient than transferring the source codes or binary codes of the interface.

4.4 Event Description

Event description describes the dynamic sides of interface, which describes the communication and control between interface and service. Event description describes the data format that needs by the semantic computation that interface transfers to the application service. The consistency of interface feedback data format could assure the device independence of the application service, improve the dialog independence of interface and application service. The absolute separation of interface and application service is often unpractical. For the data handle that needn't the interaction between interface and application, we provide a callback function lib. While interface is running, the functions in the callback function lib are called so as to fasten the speed of the interface feedback. For example, the local date function is called so as to acquire the client time.

In the event description, the description of the events base on the events that could be supported by the implement language, such as the events in the Java language.

By the method of the event handle we can see that in this method the application computation and interface presentation and the communication and control between the two are clearly defined. The description of the event handle and the final interface handle for the event has the one by one corresponding relation. So the user interface automatic generation process is predigested ulteriorly.

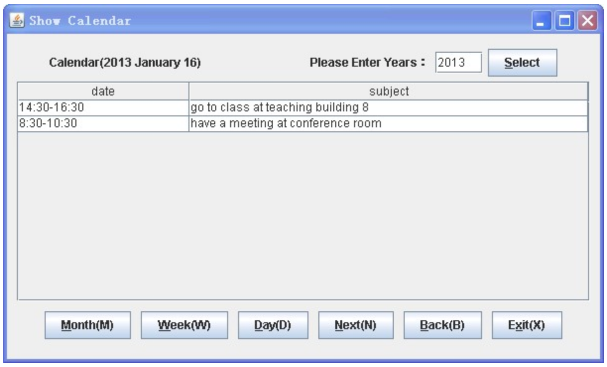


08:00:going to class at teaching building 8
10：30: having a meeting at the conference room of the administration building

Fig. 5. This figure shows smart phone version user interface

5 Case Studies

To verify that this approach can effectively support multi-device user interface development, we used this approach to build calendar service software that can add, edit, query, and show calendar. The core functions of the service and the link units located in a windows server respectively. The service communicated with the link units by the socket. We respectively developed concrete interface and event description for PC version and mobile phone version interface which used Java Swing and HTML version interface. One can access the service by the PCs and smart phones that support Java language. Figure 5, figure 6 and figure 7 are respectively the screenshot of the calendar presentation function for Java phone version, PC version and HTML version, which illuminate that the same abstract interface could realize different interface for different devices and languages. By the practically developing to verify, this approach could automatically produce the elaborate user interface for the different devices, and it is easy to study and use.



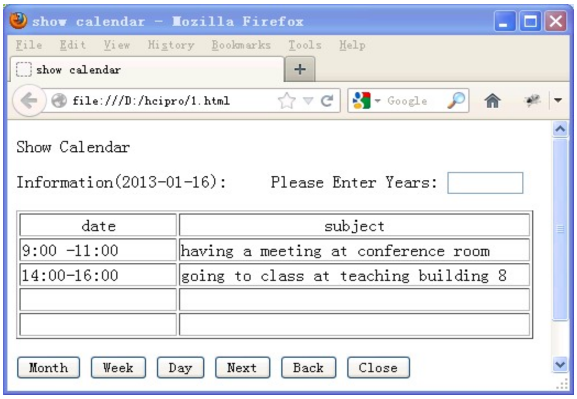
Calendar(2013 January 16)

date	subject
14:30-16:30	go to class at teaching building 8
8:30-10:30	have a meeting at conference room

Fig. 6. This figure shows the Java Swing version user interface

6 Conclusions

This paper puts forward a model-driven interactive system development approach. This approach has these advantages: 1) the classic model-driven method CAMELOAN reference framework combined with the architecture pattern, which not only make use of all advantages of user interface development that model-driven



Show Calendar

Information(2013-01-16):

date	subject
9:00 -11:00	having a meeting at conference room
14:00-16:00	going to class at teaching building 8

Fig. 7. This figure shows the HTML version user interface

Approach brings, but also the use of the architecture pattern makes the organization structure of system function components clear, deployment convenient, and is convenient to system test and evolution at the later stage. 2) In this approach the presentation and transformation of the models are straightforward and suit for all kinds of devices user interface development. 3) In this approach there isn't redundancy among the models. 4) This approach used top-down and down-top development way, which make static interface development a step wise refinement process, and in the dynamic parts of the interface application computation could better separate with the presentation of the interface. 5) This approach supports the dynamic switch interface between devices. The next work is to make this approach support more devices and languages, and to get the experience that use the approach for the large application system, so we could improve this approach.

Acknowledgments. The research is supported by the National Natural Science Foundation of China (Granted No. 61272286) and the Specialized Research Fund for the Doctoral Program of Higher Education of China (Granted No. 20126101110006).

References

1. Paterno, F., Santoro, C., Spano, L.D.: MARIA: A universal, declarative, multiple abstraction-level language for srvice-oriented applications in ubiquitous environments. ACM Transactions on Computer-Human Interaction 16(4), 19–30 (2009)
2. López-Sanz, M., Vara, J.M., Marcos, E., et al.: A Model-Driven Approach to Weave Architectural Styles into Service-Oriented Architectures. In: Proceeding MoSE+DQS 2009, pp. 53–60. ACM, New York (2011)
3. Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 16–31. Springer, Heidelberg (2005)
4. Ralph, D.H.: The abstraction—Hnk—view paradigm: using constraints to connect user interface to applications. In: Proceedings of the Conference on Human Factors in Computing Systems, California, pp. 335–342 (1992)
5. Calvary, G., et al.: A Unifying Reference Framework for multi-target user interfaces. Interacting with Computers 15(3), 289–308 (2003)
6. Saleh, E., Kamel, A.M.R., Fahmy, A.: A model driven engineering design approach for developing multi-platform user interfaces. WSEAS Transactions on Computers 9(5), 536–545 (2010)
7. Lorenz, A.: Research Directions for the Application of MVC in Ambient Computing Environments. In: Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS), pp. 336–348. ACM, New York (2010)
8. Ahmed, S., Ashraf, G.: Model-based user interface engineering with design patterns. The Journal of Systems and Software 80(8), 1408–1422 (2007)
9. Vara, J.M., Marcos, E.: A framework for model-driven development of information systems: Technical decisions and lessons learned. Journal of Systems and Software, 2368–2384 (2012)
10. Ameedeen, M.A., Bordbar, B., Anane, R.: Model interoperability via Model Driven Development. Journal of Computer and System Sciences 77(2), 332–347 (2011)
11. Izquierdo, A.L.C., Jouault, F., Cabot, J., Molina, J.G.: API2MoL: Automating the building of bridges between APIs and Model-Driven Engineering. Information and Software Technology 54(3), 257–273 (2012)
12. Agner, L.T.W., Soares, I.W., Stadzisz, P.C., Simão, J.M.: A Brazilian survey on UML and model-driven practices for embedded software development. Journal of Systems and Software 86(4), 997–1005 (2012)