Information **2015**, *6*, 122-133; doi:10.3390/info6020122

OPEN ACCESS

information

ISSN 2078-2489

www.mdpi.com/journal/information

Article

An MVC-based Intelligent Document Model Using UIML

Yunmei Shi *, Xuhong Liu †, Ning Li † and Xia Hou

Computer School, Beijing Information Science & Technology University, No 35 Beisihuan Zhonglu, Chaoyang District, Beijing 100101, China; E-Mails: cat_vs_mouse@163.com (X.L.); ningli.ok@163.com (N.L.); houxia@bistu.edu.cn (X.H.)

† These authors contributed equally to this work.

* Author to whom correspondence should be addressed; E-Mail: sym@bistu.edu.cn; Tel./Fax: +86-10-6488-4434.

Academic Editor: Willy Susilo

Received: 29 December 2014 / Accepted: 11 March 2015 /Published: 27 March 2015

Abstract: Aiming at the common problems of intelligent document platform-dependency, this paper proposes an MVC-based (Model View Controller-based) intelligent document model using UIML (User Interface Markup Language). The model is made on the basis of the previous work of our team, and the difference is that the new model separates user interface and interaction descriptions from the view component to make the intelligent document model much more independent of platform and programming language. To verify the intelligent document model, we implemented a prototype, which can support intelligent operations. The test result shows that our approach is correct. The model not only follows MVC framework, but also provides good flexibility and independence.

Keywords: intelligent document; user interface; UIML; MVC

1. Introduction

Traditionally, a document can only display content by office software according to the static description information of the document. These kinds of documents are usually called “static documents”, whose characteristic is lacking of interactivity and operability. With the development of information technology, the static documents have been unable to meet user needs. For example,

sometimes, users want office software to provide effective interaction or assistance, such as reading documents, so that users with poor vision also have the possibility of "reading" documents fluently.

Besides showing information, like traditional static documents, intelligent documents should also support dynamic operations. That means the intelligent documents can not only display document contents defined in static description information, but also can exchange data with other applications or business logic.

This paper proposes an intelligent document model based on MVC (Model-View-Controller), which uses UIML (User Interface Markup Language) to describe user interface (UI). With the help of UIML, the model separates conceptual UI (logic operation information), from actual UI and user operations, accordingly, bringing the advantages of platform-independence, reusability, and flexibility. To demonstrate the usefulness and correctness of the proposed model, this paper implements a prototype office system supporting general and dynamic document operations.

Different terms exist for intelligent documents. Besides intelligent documents, they are also called smart documents, or active documents. No matter what term is used, they have a similar meaning.

In literature [1], Heinrich presented the notion of active documents, however he just presented an ambiguous idea, rather than an explicit definition. In 2002, Professor Paolo Ciancarini *et al.* [2] definitively proposed the concept of active documents. According to their definition, an active document is composed of three parts: content, structure and behavior; in other words, active document = content + structure + behavior. In this expression, content includes data, text, images and any other static contents in a document; structure implies the representation of a document; and behavior means a method of processing a document, such as rules for defining and controlling logic processes or general computation logics. Furthermore, Chul-Ki Nam *et al.* [3] suggested another active document definition: active documents = structure + presentation + data + behavior, where structure is the logic structure of a document represented in XML; presentation is an XML Stylesheet Language document; data are information entered into a document; and behavior means a method of processing a document. These two definitions have slight differences in their expression, but they fundamentally emphasize the active behavior of processing documents. Accordingly, the most fundamental feature of active documents is the ability of processing business logic.

Microsoft first announced the concept of the smart document when releasing Office 2003, then other companies, such as Adobe, Founder, Sursen, *etc.*, put forward their opinions and definitions of the intelligent document; these companies also provided support for intelligent operations in their respective office products. However, the intelligent behaviors of these intelligent documents were utilized in different ways, such as embedding macros, scripts or plug-ins, or using DLL (Dynamic Link Library). In the former approach, dynamic operations tightly rely on specific programming languages or script languages, which means the operations cannot be shared between different platforms. As for the latter, besides depending on programming languages, it relates to specific office software. Obviously, for either approach mentioned above, there are problems, including compatibility between heterogeneous platforms and flexibility in processing intelligent operations related to processing logic.

Strictly, intelligent documents should not rely on any specific platform or programming language, and they should be able to provide personalized service for users anywhere. Aiming at this goal, different solutions have been proposed, such as adding an action description into a document model combined with XForms web service technologies. The MVC framework has also been widely applied in office

software document processing, such as HotDoc [4,5], Active Documents [6], XForms [7], InfoPath [8], and Office 2007. However, these solutions cannot solve the problems related to flexibility and independence.

Based on the research works mentioned above, our research team proposed an intelligent model based on MVC from [9] and [10]. This model separates data, presentation and dynamic operations by using MVC framework and dynamic behaviors are mainly utilized by using Web Service. In the model, the problem of platform-dependency caused by script languages or DLL is eliminated, while the view of MVC does not include the descriptions for dynamic actions, and the user interfaces for the end-users rely on some open software programs, or a specific platform, which affects the cross-platform and reusability of the model. In addition, the final UI cannot dynamically change with the application scenario, and the source code needs to be reprogrammed when the UI changes, which definitely affects the flexibility and maintainability of the model. To cope with these limitations, this paper proposes an improved model according to our team's previous research.

2. The Model

2.1. Intelligent Document Model

The intelligent document model, shown in Figure 1, also utilizes MVC framework, but is different from our previous work [9–11] in the Model component.

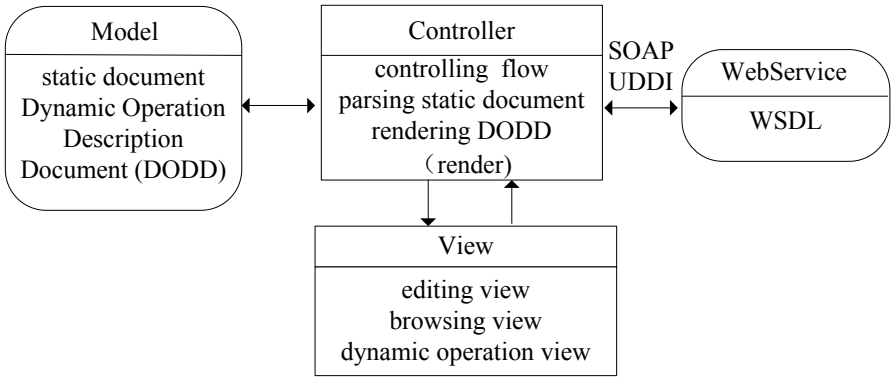

Figure 1. Intelligent Document Model based on Model-View-Controller.

(1) View is only for displaying UI.

In the document model, the only purpose of the view component is for displaying documents and operation menus in the final UI. The view has three parts: editing view, browsing view and dynamic operation view. Users can edit documents in the editing view, browse documents in browsing view, and execute various dynamic operations in dynamic operation view. The three different views share a common document model, therefore editing or modifying documents will not affect the consistency of data.

(2) The model contains almost all the description information relating to document contents, final UI and dynamic operations.

The model component in Figure 1 comprises two parts: static document and Dynamic Operation Description Document (DODD). The former provides the data and format information for the

documents, and the latter, DODD, provides the abstract representation of UI and dynamic behavior. The abstract representation does not assume any particular modality or form of delivery to the user [12]. The main objective of separating the DODD from the view component is to make the final UI independent of environment and platform, and thus improve the platform-independence of system.

In fact, the remote Web Service provides the real description of dynamic operations; the DODD is only responsible for recording the calling information of the remote services. Web Service, a kind of platform-independent distributed component, provides RPC (Remote Procedure Call) service by SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery and Integration) protocols to responding to user operation requests. Separating DODD from the view component has a great benefit: whenever the logic of processing events changes, only the DODD needs to be modified, while the Web Service component does not, therefore it is easier to reuse.

(3) The controller is mainly for passing data and messages.

The controller is used to send messages to the model, and provide the interface between the model with its associated views and the interactive user interface devices [13]. In our MVC-based document model, the controller has three effects: first, controlling the flow according to the requests from any of the three views; second, parsing the static documents to acquire content data and format information and display the content according to the specific format; and third, parsing the DODD and transfer dynamic abstract description to visual UI and the corresponding keyboard and mouse operation.

The renderer, an interpreting engine between conceptual UI descriptions and final UI, plays a key role in the above intelligent document model. Thanks to the renderer, the DODD can exist as an independent unit. Users can build various renderers for different styles of final UIs suitable for different environments and platforms. For example, the WPF (Windows Presentation Foundation) renderer displays WPF style UI with WPF toolkit, GTX (General Image Manipulation Program Tool Kit) renderer shows GTX style UI with GTX toolkit, and Java Renderer of Harmonia Incorporation transforms elements of user interface specifications into Java AWT (Abstract Window Toolkit)/Swing components.

The relationship between renderers and final UI is shown in Figure 2:

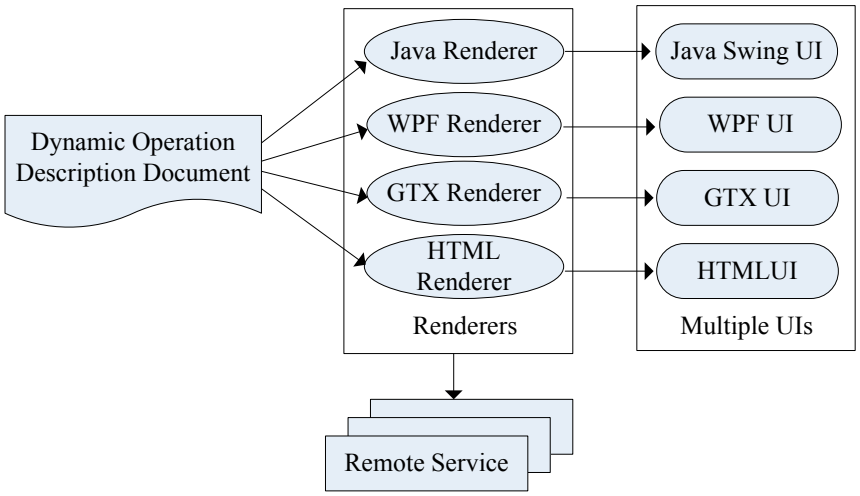

Figure 2. Renderers provide different style UI.

Our intelligent document model uses the MVC framework in order to make each component of the model as isolated as possible. The significant difference between this MVC framework and the

literature [10] is that the user interface description in our model is extracted from the view, and put into the model component. In this way, the intelligent document application based on the intelligent document model will not tightly rely on the environment or platform; consequently, the reusability of each part of the application can be greatly improved. This means that when the intelligent document application is transplanted to another platform, the only thing that needs to be done is to select an appropriate renderer, and application functions will not be affected.

2.2. Dynamic Operation Description Document

In the intelligent document model shown in the above section, user interface and behavior descriptions extracted from view are called Dynamic Operation Description Document (DODD) and they become part of the model component of MVC framework. This part can be described using User Interface Description Language (UIDL). At present, many UIDLs exists, such as XUL (XML User interface Language), MXML (Macromedia XML Markup Language), XAML (XML Application Markup Language), UIML, *etc.* After comparing the alternatives, we finally selected UIML (User Interface Markup Language) as the description language. The major reason for our choice is that UIML [14,15] is an open standard of OASIS, an organization for the advancement of structured information standards, which standards are generally being developed by a number of academic and private organizations. Furthermore, UIML provides a clear separation of content, layout and behavior; therefore it is suitable for building user interfaces for selected devices. In addition, UIML emphasizes the separation of concerns of an interactive application in such a way that moving one program from one platform to another requires little to no change [16].

Figure 3 shows the Dynamic Operation Description Document described in a UIML document and how it is being built up in relation to between final user interface and user operations.

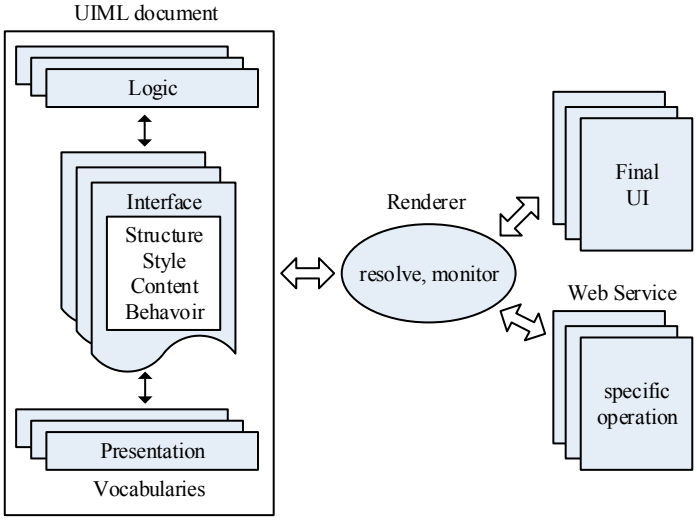

Figure 3. Dynamic Operation Description Document and its rendering.

The content contained in the UIML document in Figure 3 is just the Dynamic Operation Description Document needed in the intelligent document model, and it provides dynamic operation and abstract user interface information, including initial user interface information, event processing logic, calling

information for remote methods, and the information of user interface changes after an operation ia accomplished. Owing to only having generic elements in the UIML document, such as part, property, rule, and *etc.*, as opposed to the specific information about toolkits, the UIML document is a set of abstract concepts at high-level, therefore it is independent of target languages or platforms.

The upper right of Figure 3 is the final user interface. Users can select the user interface style they wish with the help of the renderer. The lower right is the Web Service providing remote calls, which defines the real operation logic.

Of course, a transformation from abstract concepts to their specific counterparts is needed. This is handled by a renderer located in the middle of Figure 3. As shown in Figure 3, the renderer sets up a bridge between a UIML document comprising abstract concepts and the final user interface, the operation logic. Consequently, the main task of the renderer is resolving and monitoring:

- Since the UIML document does not provide visible displaying information, the renderer needs to resolve the user interface description in it, and find the corresponding specific user interface toolkits and display them on screen.
- Monitoring means the renderer needs to notice user operations and call the remote service when necessary.

Since the UIML document follows XML specifications, and does not rely on specific languages or platforms, the content described in the UIML document are conceptual and abstract, and have no relation to any programming language or platform. Consequently, the application system based on our document model will have good advantages in flexibility and platform independence.

3. Prototype Implementation

To verify our intelligent document model, we developed a prototype system according to the model, and in the prototype we used UOF (Uniform Office Format) [17] documents, UIML-based dynamic operation description documents, Web Services, and various data sources. The system uses WPF toolkits to display its final interface, and utilizes a renderer through C#. The system architecture is shown Figure 4.

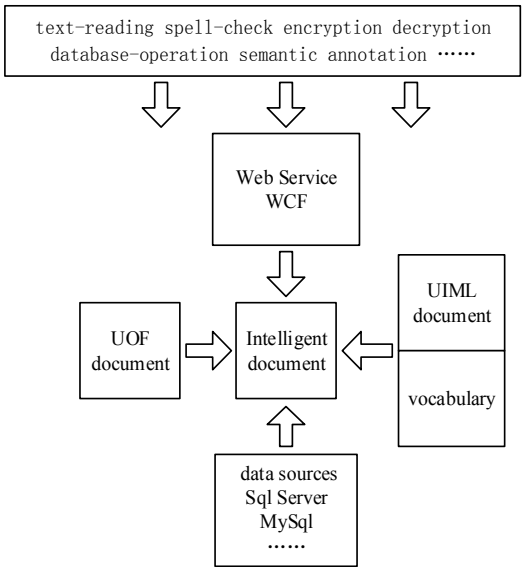

Figure 4. System architecture.

The prototype provides general operations for UOF documents, such as creating, editing, browsing, *etc.*, besides, it also supports intelligent operations: spell-check, text-reading, encryption, decryption, database operation, and semantic annotation. All six intelligent operations are realized via remote Web Service and WCF.

In the system architecture, UOF document and UIML document utilize the model component of the intelligent document model shown in Figure 1. More concretely, the UOF document corresponds to the static document, and the UIML document corresponds to the Dynamic Operation Description Document.

For the prototype, a UIML document plays a very important role for the flexibility and independence of the application, because it includes a description of the abstract user interface and event logic of user operation. The following coding fragment (Box 1) is from a UIML document of the prototype system:

```
<uiml:behavior>
    ......
    <uiml:rule>
        <uiml:condition>
          <uiml:event part-class="ButtonPressed" part-name="btnSpeaker" />
        </uiml:condition>
        <uiml:action>
          <uiml:call name="TextToSpeechService.speaker">
            <uiml:param name="text">
              <uiml:property part-name="rtb" name="Selection.Text" />
            </uiml:param>
          </uiml:call>
        </uiml:action>
    </uiml:rule>
    ......
</uiml:behavior>
```

Box 1. A UIML code fragment containing rule element.

The behavior element, containing one or more rules, describes the behavior of the dynamic operations. The rule in the above instance defines that the remote speech service should be called when the "bntSpeaker" button is pressed. However, the rule element does not support the mapping to remote service. The task is finished through another element named logic in the UIML document.

Next (Box 2) is another fragment from the same UIML document as the above instance. The d-component element in the logic section indicates the real location of speech web service described in the rule section of Box 1.

In the UIML document, the presentation element keeps the mapping from the abstract user interface descriptions to final controls. In the instance, all the mapping relationships are recorded in a vocabulary named WPF_1.0. Thus the vocabulary can be reused. In our intelligent document model, instead of being controlled by programs, both the event responses and the user interface changes are described in the UIML document, so they do not rely on the running environment. If we establish an application system

based on the intelligent document model, whenever modification of the user interface is needed, apart from UIML document, no other programs or processing logic need to be modified, which is a great convenience for system development.

```
<uiml:peers>
    <uiml:presentation base="WPF_1.0" />
    <uiml:logic>
       <uiml:d-component id="TextToSpeechService"
                         location="http://localhost:8081/TextToSpeechService/Speaker.svc">
            <uiml:d-method id="speaker" maps-to="speak">
               <uiml:d-param id="speakertext" type="string" />
            </uiml:d-method>
        </uiml:d-component>
       ......
     </uiml:logic>
</uiml:peers>
```

Box 2. A UIML code fragment containing logic element.

The system execution flow is illustrated in Figure 5, in which "shaded block 1" is for the transformation of the static document, and "shaded block 2" is for the display of the final UI and interactions with users. After producing the initial UI according to the UIML document and vocabulary, the renderer then begins to monitor user operations. When users click on the intelligent operation controls on the UI, the intelligent requests attached to the controls are also sent to the renderer at the same time. After capturing the requests, the renderer immediately searches the behavior element in the UIML document. After finding the corresponding node, the renderer will determine if the triggered condition is met, and if so, acquire the intelligent operation steps and call methods of remote services. After accomplishing the execution of the intelligent operations, the renderer will update the intelligent operation UI according to the information changes of the UI recorded in the UIML document.

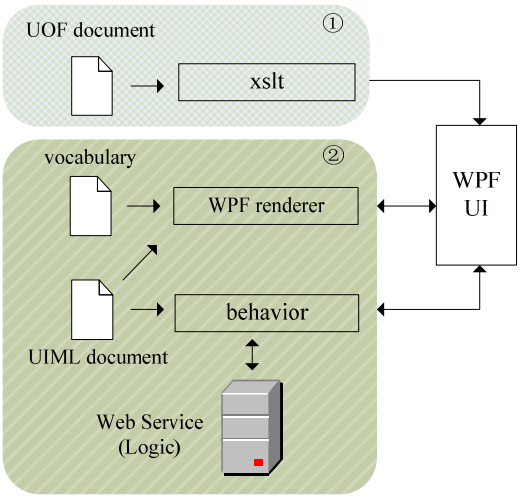

Figure 5. Processing flow of the intelligent document application.

Figure 6 is the initial interface of our prototype. The left buttons in the figure are for general operation, for example opening, editing UOF documents, *etc.*, while the right buttons are for intelligent operations. When clicking on the right buttons, the interface will be switched to the intelligent operation interface shown in Figure 7.

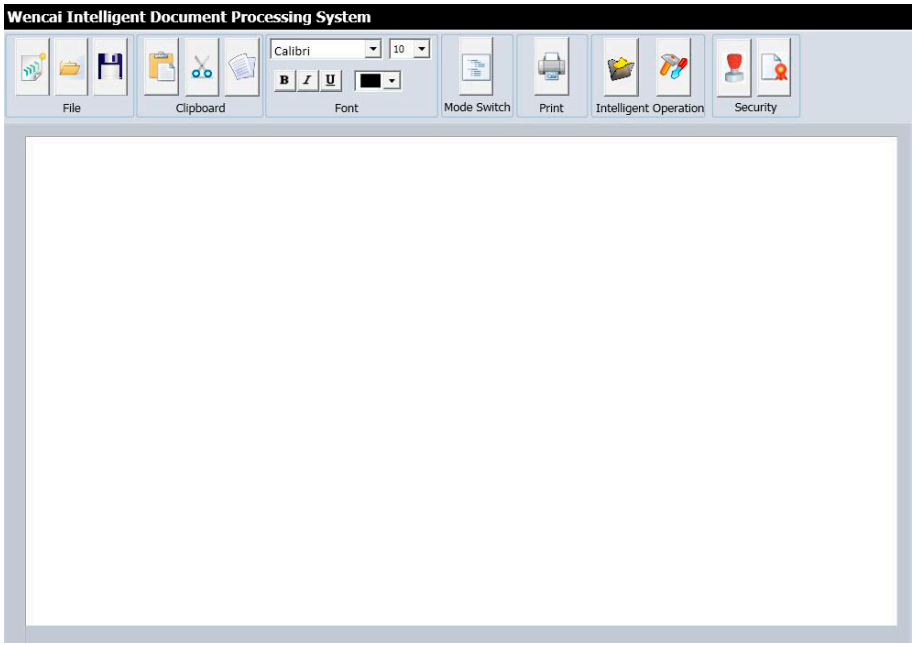

Figure 6. The initial view for browsing documents.

The intelligent operation interface includes six intelligent buttons: spell-check button, text-reading button, encryption button, decryption button, database-query button, and semantic annotation button.

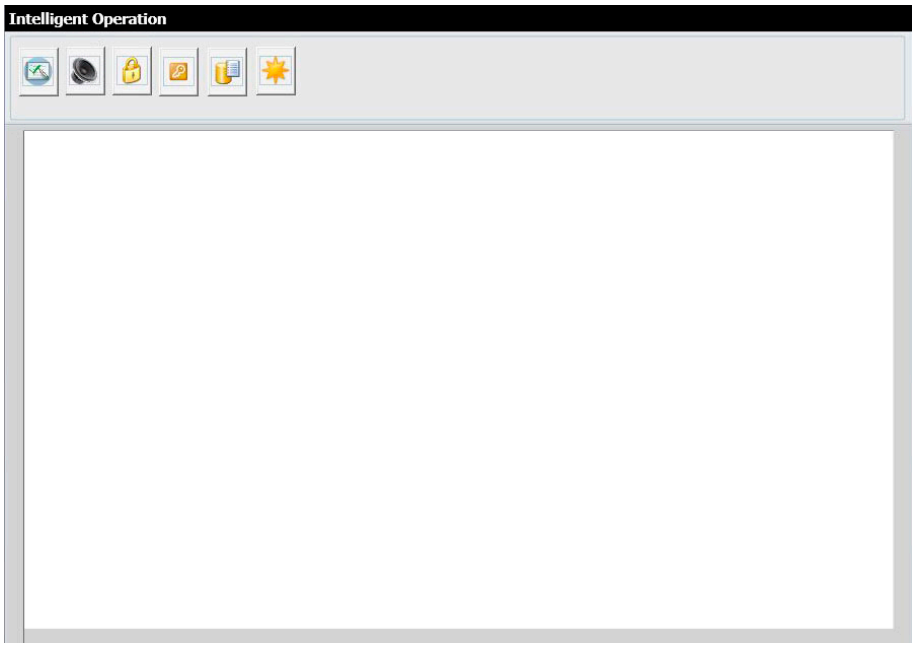

Figure 7. Intelligent operation view containing six intelligent controls.

Aiming at the system functions, especially the six intelligent operations mentioned above, we tested a total of 23 cases on the Windows XP Professional SP 3 platform. The test cases can be classified into three groups: test cases for general operations, test cases for switching between different views, and test cases for intelligent operations.

In the first group tests, we tested the frequently used operation functions, for example creating, opening, saving, and printing a UOF document. In the second group, we tested switching between the browsing view and the editing view. In the final group, we tested all the intelligent operations utilized in the prototype. All the tests were correctly executed and test results meet our expectations.

For this paper, we only developed one renderer for WPF toolkits. If we want to transplant the system to other platforms that are not suitable for WPF controls, or change WPF toolkits to other toolkits that users wish, the only thing that needs to be done is developing the appropriate renderer. This task is a lower cost compared to rebuilding a new system on a different platform that has the same functions. If we want to increase or decrease controls on the final UI, or change the UI layout, we simply modify the UIML document.

4. Conclusions and Future Work

In this paper, we have presented a UIML-based intelligent document model for separating data, view and behavior description from the final UI. The advantages of the model can be summarized as following:

- Standardized UI description: the abstract UI is described using standardized user interface description language.
- Platform-independence: the programs are separated from the UI description, and the system interaction with users is separated from the particular platform. So the system set up, according to the intelligent document model, is easy to move from one platform to another, for example from Windows to Linux.
- Language-independence: no matter what advanced programming language used, the described abstract UIs can be utilized.
- Reusability: some components of the model can be reused, for example both the vocabulary and remote procedure call.
- Reduced complexity of development: the separation of programs and UI makes programmers free from the complicated UI development, and pay more attention to programming other functions.
- Improved flexibility: as the function of the application has nothing to with the UI, the application skin, the final UI presented to end-users, can be easily changed.

The paper attempts to describe an intelligent document model, which still has some deficiencies. For example, in the intelligent document model, we use “absolute positioning” for the UI controls display, so if the device changes, the display layout defined in the UIML document needs to be modified, which reduces the model’s independence. If we want a UI description that can be applied to any device, it needs to be improved. So in future research, we intend to improve our model by way of controlling the view completely through the model component of our MVC intelligent model; thus making the view independent of the specific platform, and able to be correctly shown on different devices.

Acknowledgments

This work originated in the Beijing Municipal Education Commission, under the grant (KM201511232012) and project "key technology research on UIML-based intelligent document" supported by the Opening Project of State Key Laboratory of Digital Publishing Technology. We would like to thank the funding agency for supporting our research.

Author Contributions

The research scheme was mainly designed by Ning Li. Yunmei Shi, Xuhong Liu, and Xia Hou performed the research and analyzed the data. The paper was mainly written by Yunmei Shi. All authors have read and approved the final manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

References and Notes

1. Heinrich, E.; Maurer, H.A. Active documents: Concept, implementation and applications. *J. Univ. Comput. Sci.* **2000**, *6*, 1197–1202.
2. Ciancarini, P.; Tolksdorf, R.; Zambonelli, F. A survey of coordination middleware for XML-centric applications. *Knowl. Eng. Rev.* **2002**, *17*, 389–405.
3. Nam, C.K.; Jang, G.S.; Bae, J.H.J. An XML-based active document for intelligent web applications. *Expert Syst. Appl.* **2003**, *25*, 165–176.
4. Buchner, J.; Fehnl, T.; Kunstmann, T. HotDoc—A Flexible Framework for Spatial Composition. In Proceedings of the 1997 IEEE Symposium On Visual Languages, Isle of Capri, Italy, 23–26 September 1997; pp. 92–100.
5. Buchner, J. HotDoc: A framework for compound documents. *ACM Comput. Surv.* **2000**, *32*, doi:10.1145/351936.351969.
6. Bompani, L.; Ciancarini, P.; Vitali, F. Active documents in XML. *SIGWEB Newsl.* **1999**, *8*, 27–31.
7. XForms Current Status. Available online: http://www.w3.org/standards/techs/xforms#w3c_all (accessed on 11 March 2015).
8. Microsoft InfoPath. Available online: http://en.wikipedia.org/wiki/InfoPath (accessed on 11 March 2015).
9. Wen, J.; Li, N. A Research on Expression of Document Operation. *Comput. Sci.* **2007**, *34*, 375–376. (In Chinese)
10. Li, W.; Li, N.; Wen, J. A Model for Dynamic Document Operation and Presentation. *Appl. Res. Comput.* **2010**, *27*, 827–829. (In Chinese)
11. Fan, K.; Li, N.; Wu, Q.; Liu, X. Design and Implementation of an On-Line Office Document Processing Tool Based on XForms. *Comput. Eng. Sci.* **2011**, *33*, 180–185. (In Chinese)
12. Trewin, S.; Zimmermann, G.; Vanderheiden, G. Abstract user interface representations: How well do they support universal access? In Proceedings of the 2003 Conference On Universal Usability, Vancouver, BC, Canada, 10–11 November 2003; pp. 77–84.

13. Krasner, G.E.; Pope, S.T. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *J. Object Oriented Program.* **1988**, *1*, 26–49.
14. User Interface Markup Language (UIML) Committee draft. Available online: http://www.oasis-open.org/committees/download.php/28457/uiml-4.0-cd01.pdf (accessed on 11 March 2015).
15. Abrams, M.; Helms, J. *User Interface Markup Language (UIML) Specification version 3.0*; Harmonia Inc.: San Diego, CA, USA, 2002.
16. Ali, M.F.; Pérez-Quiñones, M.A.; Abrams, M.; Shell, E. Building Multi-Platform User Interfaces with UIML. In Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, Valenciennes, France, 15–17 May 2002; pp. 255–266.
17. Uniform Office Format. Available online: http://en.wikipedia.org/wiki/Uniform_Office_Format (accessed on 11 March 2015).