Interactive Office Documents: A New Face for Web 2.0 Applications

John M. Boyer
IBM Victoria Software Lab
4396 West Saanich Road
Victoria, BC, Canada V8Z 3E9
boyerj@ca.ibm.com

ABSTRACT

As the world wide web transforms from a vehicle of information dissemination and e-commerce transactions into a writable nexus of human collaboration, the Web 2.0 technologies at the forefront of the tranformation may be seen as special cases of a more general shift in the conceptual application model of the web. This paper recognizes the conceptual transition and explores the connections to a new class of *interactive* office documents that become possible by tighter integration of the Open Document Format with the W3C's next generation web forms technology (XForms). The connections transcend simple provisioning of office document editing and persistence capabilities on the web. Rather, the advantages of office documents as self-contained entities that flow through a collaborative network or business process are combined with web application qualities such as intelligent behavioral interaction, in-process web service access, and control of server submission content. An office document mashup called 'Dual Forms' is presented to demonstrate the feasibility of office document centric web applications.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*Markup languages, Standards*; H.1.2 [**Information Systems**]: User/Machine Systems—*Human information processing*; K.4.4 [**Computers and Society**]: Electronic Commerce—*security*

General Terms

Standardization, Languages, Security, Legal Aspects

Keywords

Office Document, Business Process, ODF, web service, SOA, XForms, XML Signature, user interaction

1. INTRODUCTION

The first 'killer application' of the personal computer revolution was the word processor, which transferred a certain amount of computing power to the hands of the people by giving them the power to create and, with a little effort, share content without the aid of computer programmers. But what sealed the trajectory of the personal computer revolution was proof that it was more than just a one-trick pony. The second killer application, the spreadsheet, enabled a large class of end-users to declaratively express useful computer applications, again without the aid of computer programmers. This is a progression that some have called *process democratization* [20].

We are now seeing an analogous revolution play itself out on the world wide web. The web itself is transitioning from a medium that can only be configured by web developers to one whose content is controllable by the very consumers of that content. This writable web medium, which has been called Web 2.0, essentially began with the blog [4]. However, the Web 2.0 analog of the word processor is really the wiki [14], since it allows end-users to collaboratively create and update shared web content without the aid of web developers. Similarly, the early Web 2.0 analog of the spreadsheet is an XML markup called XForms [8], which enables a large class of web content authors to declaratively express useful web applications [9], either without the aid of web developers or with their efforts focused on connecting the results of completed applications to back-end business processes [13].

To illustrate these analogies, the transition of the conceptual model for Web 2.0 applications is presented in Figure 1. Under the current 3-tier model, the web application programmer uses custom-built middle tier code for two reasons:

- to backfill for an underpowered client tier by driving user interaction with server code
- to assemble and perform transactions with the rigid APIs of productized server tier applications

By comparison, Web 2.0 applications do not tend to involve a middle tier web application developer. On the client-side, this is due to the rise of web browser infrastructure technologies, e.g. Javascript, DOM and AJAX, that are capable of meeting the end-user interactivity and live update requirements of the web application without round-tripping to *custom code* on the middle tier. On the server-side, the early Web 2.0 applications, such as blogs and wikis, have replaced the rest of the middle tier with specific code for the application. However, the full generalization of the Web 2.0 application model can be realized on the server-side due to the trend in productized server tier applications to open their API interfaces as web services or to allow manipulation of persistent storage by ATOM publishing protocol [19]. Hence, the Web 2.0 client tier can communicate directly with

DocEng'08, September 16–19, 2008, São Paulo, Brazil.

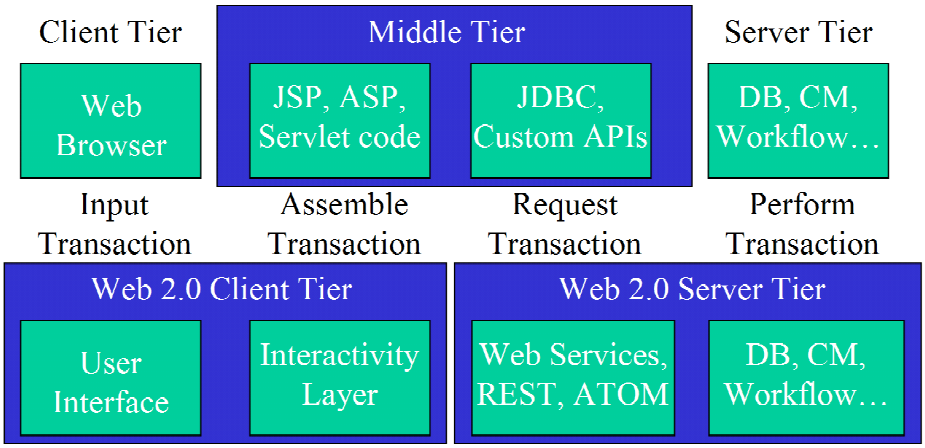


Figure 1: The Transition from the current 3-tier web application model to the 2-tier conceptual model for Web 2.0 Applications. The server tier is augmented by web services or ATOM publishing protocol to allow direct communication and interaction with the client tier. The client tier is sufficiently powerful to drive end-user interactivity for the application and to communicate with the server tier.

the Web 2.0 server tier to gain access to persistent data storage, workflow and business process services.

This is the maturing web context in which office documents can come to serve as a new face for Web 2.0 applications. Even at the simplest level, the office document provides a familiar user interface, so it is natural to consider the office document as a metaphor for the client side of a web application. Yet, the office document makes sense for more than just the ephemeral user interface for back-end content. By *persisting* the results of user interaction in an office document format that is based on XML [12], several of the more well-known benefits can be realized, including:

- ability to locally save and reload or email the document
- compatibility with the offline document processor for disconnected and semi-connected users
- simplified data mining and content reusability (e.g. presentation slide re-use, or spreadsheet content re-use in a presentation slide or text document pie chart)

Aside from the above core benefits of a document-centric web application mentioned above, there are three advanced benefits that comprise the major focus of this paper:

- ability to define, within the document, intelligent behaviors for user interaction and web service access
- ability to affix XML Signatures [16] to the document to make a legally binding agreement for the transaction represented by the document
- support for document-centric business processes, workflows and activity-centric systems, not just document persist-and-share systems

To realize all of these benefits, this paper explores the new open standard XML markup for office documents, the open document format (ODF) [15]. In particular, this paper focuses on how the three advanced benefits above can be realized through better exploitation of the underlying XForms model that has been incorporated into the ODF standard. Furthermore, while XForms provides more features to ODF than is generally known, ODF references the latest version of XForms, so still more features will be available to ODF when the current XForms 1.1 Candidate Recommendation [8] becomes a W3C Recommendation later this year, especially in the area of XForms submissions for connecting to web services and performing full document submissions.

After a discussion of related work in Section 2, a number of features of the XForms model that are currently available to ODF are introduced in Section 3, and Section 4 provides an overview of the document structure of ODF combined with XForms, especially describing how the ODF presentation layer binds to data in the XForms model. Section 5 describes how those XForms user interface bindings improve interactivity by helping end-users to correctly 'fill in the blanks' of complex agreement documents. The section also discusses how to improve user experience with a wizard-like interaction modality that orchestrates the client-side fill experience of a complex document. Section 6 describes how the new XForms 1.1 submissions can be used to connect a running ODF document to an SOA (service oriented architecture), to RSS and ATOM feeds and ATOM publishing protocol, and how the entire ODF office document can be connected to the business process. Section 7 describes how digital signatures can be applied to ODF documents through the XForms data layer to create legally binding e-contracts and e-agreements. Section 8 presents a prototype application created to illustrate the value of the many possibilities that arise from connecting interactive office documents to the business process. Section 9 presents concluding remarks and future work.

2. RELATED WORK

A number of recently proposed systems use documents to drive *only the design phase* of web applications. The system in [23] automatically generates classic HTML 4 forms based on structure of data-centric XML vocabularies. The work in [23] is focused on generating a traditional web application, not in changing the information assets over which web applications operate. The system is extended in [24] to connect

the XML data collected into server workflow systems. In [1], the design experience is divided into separate concerns of content, navigation and presentation, and each is represented by a model document. The web application is created by performing XSLT transformations on the model documents to generate scripts, stylesheets and presentation content. Though the model documents separate the concerns, they are not kept in a single file used to drive the *run-time* experience, so the advantages of a document-centric web application described in Section 1 are not achieved.

The earliest XML document format to combine dynamic user interface behaviors, digital signatures, and support for the document as a mobile agent in the business process was XFDL [3, 10] (extensible forms description language). Malloy [26] describes an intelligent document as being more like an application than a traditional document. It is essentially *an instance of* an application comprised of a content or data layer, a layer for logic and behavioral rules, a presentation layer, and a layer of metadata. Glushko and McGrath [18] note the importance of a layered document structure to the document engineering analyses needed to describe business processes that produce and consume documents, and the integration of XFDL with XForms reported in [6] achieves the layered intelligent document structure for precision electronic forms applications that capture *structured* data.

In light of the layered approach advocated above, it is worthwhile comparing other recently reported methods with the approach to document dynamism in XForms [8]. In [25], an XML data calculation is achieved by attaching an attribute bearing an XPath expression to the element whose content is being calculated. The XPath is re-evaluated whenever a computationally referenced element is changed at runtime. The approach in [25] mixes the data layer with the logic rule layer, which may not even be permitted by the data schema. Also, it is not capable of computing attribute values. Similarly, [28] presents a template process for generating presentation level markup configured according to parameters representd as XPath variables. Aspects of the presentation markup requiring configurability invoke a 'calc' function in lieu of a literal value. This is analogous to the method used by XFDL to implement declarative presentation logic [3, 6, 10], but it requires participation of the presentation language processor. The method of [28] assumes the ability to modify the presentation language schema and the ability to inject XPath evaluation and an update processing model into the presentation language processing. From a standardization perspective, these assumptions are difficult because they hinder wide implementation and black box reuse of existing presentation engines. Thus, the layered document approach used in XForms avoids certain difficulties of the above cited works.

The conceptual separation of layers promoted by XForms is also what allows it to satisfy its *multimodality* requirement. XForms was designed to make it easy to define the core XML data process asset and then 'skin' the asset with mulitiple presentation layers according to system/user needs. This allows systems such as those described in [13, 21] to deploy the same application asset to a desktop browser, a mobile device, or a device for the sight-impaired. Moreover, it allows the same core asset to be presented in multiple official languages of a country so that a government can ensure equality of application behavior regardless of the language of interaction selected by the citizen.

A side-effect of this multimodal design is that new presentation languages (host languages) for XForms can be conceived beyond traditional web markup languages. One such markup language is the open document format (ODF) [15], the open standard XML markup language for office documents. Toward the direction of using office documents as a familiar user interface metaphor for web applications, Gibson [17] suggests that it can help provide a solution for the problem of web application accessibility. Also, Quint and Vatton [27] advocate that web browsers should be able to write web pages, not just browse them, and Di Iorio and Vitali [22] describe a simple word processor interface for editing web pages. However, this paper advocates the use of the office document as more than just a user interface metaphor, but rather as the underlying unit of information in web applications and business processes. In particular, the rich Web 2.0 client tier interactivity and Web 2.0 server tier communications can be achieved by the ODF office document format via its integration with XForms 1.1, as discussed in the sections below.

Finally, the flexibility of deployment models noted in [13] is also feasible for interactive office documents. The runtime processing can be offered by client-side software or by server-side conversion to the HTML, Javascript and AJAX code that is natively understood by web browsers. In fact, the logical client-side interactivity expressed in XForms can be executed on the physical client-side in the web browser using a Javascript and AJAX library such as the Ubiquity XForms processor [2].

3. INTRODUCING THE XFORMS MODEL

The backbone of XForms is the processing model that is defined for XML data within the document. A document can have one or more *instances* of XML data, each of which is handled as if it were a separate in-memory document object model (DOM) during execution of the document. The name *instance* comes from the possibility that the XML data is an instance of an XML schema, though an XForms instance is not required to be associated with a schema.

The XForms model augments the XML instance data with *model item properties* such as `type`, `readonly`, `relevant`, `required`, and `constraint`. These model item properties associate run-time metadata with nodes of XML data, and they are attached to nodes of XML data using using XPath expressions. The values of most of the model item properties are also determined using XPath expressions. This allows the form author to specify the model item property value by a formula that is *automatically* re-evaluated whenever dependent nodes of XML data are changed. Perhaps the most important model item property of an XML data node is its intrinsic value, which may be set by a `calculate` formula. Figure 2 gives a markup example of a small XForms model.

A critical part of the XForms model is its *submission* capability. A `model` can contain any number of `submission` elements, each of which can respond to events that perform further actions, such as data mutations or further submissions. The result of a submission can replace the entire document containing the XForms, or it can replace some data within the XForms model, or it can be ignored. Figure 3 provides an example of XForms submission that submits a term to a dictionary service and places the definition text obtained from the service into a node of the data without replacing the entire containing document.

```
<xf:model xmlns:xf="http://www.w3.org/2002/xforms"
          functions="power">
    <xf:instance xmlns="">
        <Pythagorus>
            <a>5</a>
            <b>12</b>
            <c>13</a>
        </Pythagorus>
    </xf:instance>

    <xf:bind nodeset="a | b" required="true()"/>

    <xf:bind nodeset="c"
             calculate="power(../a*../a+../b*../b, 0.5)"/>
</xf:model>
```

Figure 2: An XForms model containing one instance and a few model item property (MIP) definitions. The `bind` element expresses MIPs using attributes to hold the XPath expressions. The `nodeset` attribute indicates the node or nodes to which the MIP values are associated. The first `bind` attaches a `true` value for the `required` property to the union of nodes `a` and `b`. The second `bind` calculates the value of `c` according to the well-known hypotenuse length formula.

```
<xf:model xmlns:xf="http://www.w3.org/2002/xforms">
    <xf:instance xmlns="">
        <search>
            <term>omnify</term>
            <meanings> ... </meanings>
        </search>
    </xf:instance>

    <xf:submission ref="term" target="meanings"
           replace="text" method="get"
           resource="http://dictionary.com/service"/>
</xf:model>
```

Figure 3: An XForms model can perform submissions to a server while the document is running, and the results can be placed back into the document. For example, the meaning of a word could be searched using an online service.

4. ODF INTEGRATED WITH XFORMS

ODF integrates the XForms model into its `form` element, which also includes a set of abstract form control elements such as `form:text` that select XML data to operate upon and set other basic user interface properties. ODF exposes its abstract form controls to the free-flowing presentation layer markup using the `draw:control` element. Figure 4 depicts the main layers in the integration of ODF and XForms.

The relationship between the XML data nodes, abstract form controls and presentation layer elements is handled mostly by ID references. From Section 3, an XForms model can contain any number of `xf:bind` elements, each of which indicates one or more nodes using a `nodeset` attribute. An `xf:bind` can also have an `id` attribute, which makes it a named site for a set of nodes that can be referenced by the `id` attribute value. An ODF form control uses an `xf:bind` attribute to indicate an `xf:bind` element by ID reference. Similarly, each ODF form control has a `form:id` attribute that allows the presentation layer `draw:control` element to refer to it with a `draw:control` attribute. Figure 5 puts the relevant markup components together to illustrate the ID referencing mechanism that binds the ODF layers together.

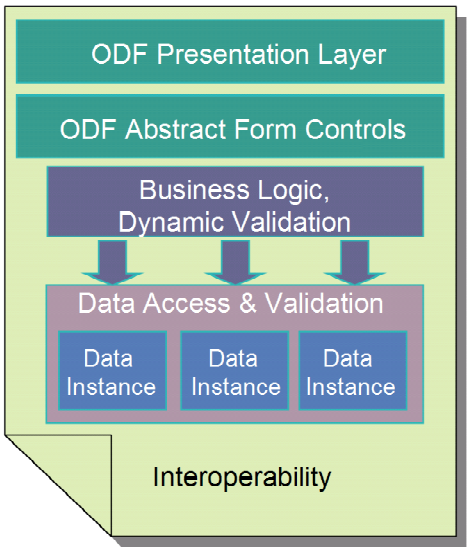


Figure 4: The document structure of the ODF integration of XForms. The basis in XML means interoperability from the ground up, which is why XForms was able to be incorporated into ODF.

5. INTERACTIVE OFFICE DOCUMENTS

The `xf:bind` attribute in XForms has more meaning than just connecting an ODF form control to an XForms bind site. The `xf:bind` attribute expresses a user interface binding, which has several implications for the XForms model processor, which must:

- expose the node value and model item property values to the bound form control
- dispatch value and model item property change notification events to the bound form control

If the XForms model binds a `type` or `constraint` model item property (MIP) to a node, and the user enters incorrect data content, then the form control must prominently indicate that there is an error, and the XForms model processor must dispatch an `xforms-invalid` event to the element containing the user interface binding attribute. XForms authors have the option of hooking this event and performing an action script, such as raising a `message` to help the user understand how to correct the error or perhaps executing a `setvalue` to fix the error under certain conditions.

If the XForms model binds a `readonly` MIP to a node, and the value is *true*, then the user is not allowed to modify the data. Note that the readonly MIP has an inheritance rule, so it is easy to make all nodes in a DOM subtree readonly by just setting the subtree root to readonly. This can make entire sets of bound form controls behave as if they were readonly. Form authors (as well as implementations

```
<office:document-content ... >
    ...
    <office:body>
        <office:text>
            <office:forms ... >
                <xf:model ...>
                    <xf:instance ...>
                        <name>Wanda</name>
                    </xf:instance>

                    <xf:bind id="Data_Name"
                             nodeset="/name"/>
                </xf:model>

                <form:form ...>
                    <form:text xf:bind="Data_Name"
                               form:id="Ctl_Name" ... >
                        <form:properties>
                            ...
                        </form:properties>
                    </form:text>
                    ...
                </form:form>
            </office:forms ... >
            ...
            <text:p text:style-name="P20">
                <text:span>Name: </text:span>
                <draw:control svg:x="2in" svg:y="1.5in"
                        draw:control="Ctl_Name" .../>
            </text:p>
            ...
        </office:text>
    </office:body>
</office:document-content>
```

Figure 5: The ID referencing mechanism that connects XML data to the ODF presentation layer. The `xf:bind` associates an ID with a data node. The ODF form control element associates an ID for the control with the ID for the data so the ODF presentation layer element can present the data by referencing the ID of the associated ODF form control.

like an ODF processor) can detect changes of state after initialization by listening for `xforms-readonly` and `xforms-readwrite` events dispatched by the XForms model.

If the XForms model binds a `relevant` MIP to a node, and the value is *false*, then the form control must be either hidden or disabled. This MIP also has an inheritance rule that makes it easy to affect many form controls bound to nodes in a subtree by setting the subtree root node's relevance. For this MIP, it is especially important to note that the XForms model supports dynamic recalculation of MIPs because conditional relevance can be used to choreograph wizard-like behaviors that take a user through a step-by-step process for completing a complex fill experience.

Although it is beneficial to the user to simplify the user interaction with a stepwise wizard experience, a common concern about using relevance to choreograph the user experience is that the non-relevant data is also unavailable to data submissions. It would be beneficial to use relevance for user experience choreography without having any side effects on the data that can be submitted to a server or service. One answer to this problem is to simply hook the `xforms-submit` event in the model and use the `insert` action to copy the data to a second instance that has no relevance rule bindings. Figure 6 contains illustrative XForms 1.1 markup for this method.

```
<xf:model xmlns:xf="http://www.w3.org/2002/xforms">
    <xf:instance xmlns="" id="X">
        <data>
            <stepA>...</stepA>
            <stepB>...</stepB>
            ...
        </data>
    </xf:instance>

    <xf:instance xmlns="" id="Y">
        <data>...</data>
    </xf:instance>

    <xf:bind nodeset="instance('X')/stepA"
             relevant="condition A"/>
    <xf:bind nodeset="instance('X')/stepB"
             relevant="condition B"/>

    <xf:submission ref="instance('Y')" ...>
        <xf:insert ev:event="xforms-submit"
                   nodeset="instance('Y')"
                   origin="instance('X')"/>
    </xf:submission>
</xf:model>
```

Figure 6: An XForms model that uses relevance to implement a step-by-step wizard. The user interface controls bind to instance X. At submission time, the data is copied to instance Y, a fully relevant location.

Yet another answer to the above problem, though, is to stop doing data submissions and instead use XForms submission to connect the document to the business process.

6. XFORMS 1.1 SUBMISSIONS FOR DOCUMENT-CENTRIC WEB 2.0 APPLICATIONS

XForms 1.1 submissions have been substantially improved to allow both document-centric web applications and Web 2.0 interactions during document run-time. This will allow an ODF document to submit its entire content, rather than just data, to the back-end business process. It will also allow an ODF document to consume SOAP-based web services and ATOM services exposed by the Web 2.0 server tier described in Section 1.

The rationale for full document submission is simplified user experience during collaboration. Directly supporting collaborative content creation significantly reduces inefficiencies relative to manual collaboration based only on change tracking, saving to disk, emailing, and manually pushing only completed document into business process systems. Figure 7 is an example system diagram in which intelligent documents "know" what process they belong to and how to get themselves from each collaborator back to the server without need of local saving or email.

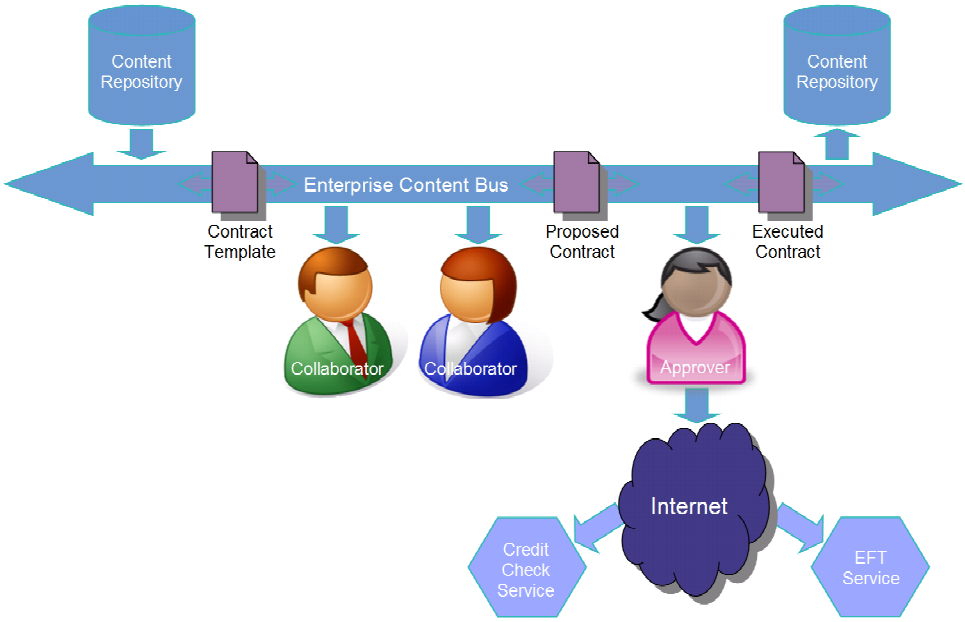


Figure 7: The Intelligent Document Programming Model goes beyond the traditional usage paradigm of office documents to get at some of the underlying reasons *why* people are using office documents. Users are able to retrieve an office document on which they are collaborating, update it, and return it to the enterprise content bus on the server-side so it can be accessed by other users in the business process workflow. As the document proceeds through the business process, users can interact with it manually, and they can also access web services to enhance the completion experience.

To transport an ODF document to a server, it is possible to use a feature of XForms 1.1 submission in a novel way. A new event called `xforms-submit-serialize` has been added to allow form authors greater control over the server post data. For example, a form author can submit text rather than XML to the server. The XForms 1.1 submission now also supports a `serialization` attribute that allows the form author to indicate the content type of the alternate serialization, especially if it is not XML content.

This feature can also be used to allow *host document processors* to control the content uploaded by an XForms submission. Since `xforms-submit-serialize` bubbles up to the root of the document, an ODF processor can be defined to listen for the event, detect if an ODF content type appears in the value of the `serialization` attribute of the event target `submission` element, and if so, redefine the XForms submit serialization to contain the entire ODF document. This will allow the document to return itself to the server-side business process to participate in further workflow steps or be stored in a content repository as a completed document. Figure 8 show a markup example of full ODF office document submission to the server.

Figure 7 also illustrates the possibility of end-user interaction with the Web 2.0 server tier during document run-time. The specific example shows consumption of credit check and electronic funds transfer services. These services may produce record locator data that must be stored in the document before it is finalized. Other services such as database search results can be invoked throughout the collaborative fill experience to reduce typing and data entry errors.

RSS and ATOM feeds obtained by the 'get' or 'post' HTTP method can be consumed by XForms 1.0 applications. XForms

```
<xf:submission
    resource="http://www.example.org/someServerScript..."
    serialization="application/vnd.oasis.opendocument.text;
                   content-encoding=base64"
    ... />
```

Figure 8: An XForms submission whose serialization can be defined by an ODF processor to send the entire ODF document to the server, connecting it to the business process. The base-64 encoding is used to account for the possible binary result of the ODF packaging format.

1.1 adds the 'put' and 'delete' methods so that all CRUD operations can be performed through ATOM publishing directly from an XForm with no middle tier coding. XForms 1.1 also adds full SOA enablement for both REST services and SOAP-based web services. REST services are supported due to the following additions to submissions:

- control of submission method using a static or computationally derived NCName in the `method` attribute
- control of submission headers using any number of `header` child elements of submission
- ability to invoke functions `hmac`, `digest`, and `random` in combination with setting headers.
- access to headers, return codes and other metadata about the submission result in the context information of the `xforms-submit-done` event

Finally, web services can be invoked via the HTTP SOAP binding for XForms submissions defined in Section 11.11 of [8]. The section includes a complete markup example written by the author for a request-response web service, including separating the core instance data from the SOAP request and response envelopes. The key to the request, though, is simply setting the `mediatype` attribute to `application/soap+xml` plus the `action` MIME parameter to specify the SOAPAction. While this is the information needed for a SOAP 1.2 service, the XForms submission processor automatically converts to the settings needed for SOAP 1.1 if the submission data is rooted by a SOAP envelope element in the SOAP 1.1 namespace.

Although access to web services can enhance the user experience, at some point the user interaction is completed and the user would like to submit the completed document to the server-side business process. In some systems designed for highly regulated industries, the last step of user interaction before full document submission may be to affix a digital signature to ensure that no malicious third party can corrupt the document as completed by the user. The next section is focused on digital signature security for office documents.

7. XML SIGNATURES IN ODF

Digital signatures are a natural part of the theme of connecting office documents to the business process in order to meet the auditability and security needs associated with execution of legally binding contracts and agreements.

Since ODF is an XML format, it is reasonable to add digital signature support to ODF by integrating with the W3C standard for XML Signatures [16]. Note that the XML Signatures standard defines an XML markup for expressing digital signatures that can sign any number of resources, whether or not they are encoded as XML. This is ideal for ODF since the ODF packaging format [15] allows a single ODF document to be comprised of any number of resources, including binary files for embedded images.

Since digital signatures are a kind of structured input, it is reasonable to integrate them into ODF through XForms. A framework for integrating XML Signatures with XForms appears in [7]. The design presented in [7] is predominantly concerned with the mechanics of activating digital signature generation and validation in such a way that host language processors can augment these operations with visual security algorithms described in [5]. These algorithms, or other alternative augmentations, are useful additions to document security in advanced multiple signer scenarios.

However, the main consideration for a basic integration of XML signatures with ODF via XForms is to recall that XForms treats the data layer as a set of separate DOMs from the main document during run-time. Therefore, a *same-document URI-reference* as defined in [16] refers to the root of the containing XForms instance, not the `office:document-content` element. In order to sign the ODF content, a `dsig:Reference` must use a package-relative URI to indicate the *content.xml* file.

This leads to an interesting problem because the signature is still being added indirectly to the *content.xml* file via adding it to the XForms instance data in the ODF content. Such a signature is defined by [16] to be an *enveloped signature* because the signature is enveloped by the content it signs. An enveloped signature must remove itself from the content it signs because otherwise core signature generation will modify part of the content after it has been digested (the `dsig:SignatureValue` is empty when the content containing the signature is digested).

Although the ODF content envelopes the XML signature, the enveloped signature transform cannot remove the signature from the ODF content being signed because enveloped signature transforms and the `here()` function only work on same-document URI-references. This is all for the best, though, because the original XPath filtering method in [16] is too slow to use on office documents, so the typical implementation of the enveloped signature transform is likely to be too slow as well. Instead, as exemplified in Figure 9, the signature must be subtracted using XPath Filter 2.0 [11].

```
<office:document-content ... >
  ...
  <xf:instance ...>
    <my:data >
      ...
      <ds:Signature id="X">
        ...
        <ds:Reference URI="content.xml">
          <ds:Transforms>
            <ds:Transform
              Algorithm=".../xmldsig-filter2"
              xmlns:f2=".../xmldsig-filter2">
              <f2:XPath Filter="subtract">
                //ds:Signature[id='X']
              </f2:XPath>
            </ds:Transform>
          </ds:Transforms>
        </ds:Reference>
        ...
      </ds:Signature>
    </my:data>
  </xf:instance>
  ...
</office:document-content>
```

Figure 9: The pertinent parts of an XML Signature that signs ODF content while subtracting itself as an enveloped signature.

The basic ability to add an enveloped XML signature to an office document lends directly to solving more complicated signing scenarios that arise in highly regulated industries. Often, such documents need more than one signer to reflect the due diligence of an approval process. For example, consider the signer/approver pattern. When the first signer affixes a signature, the signed content becomes immutable, which means the approver cannot affix a second signature without invalidating the signature of the first signer. The solution is to use another XPath filter subtraction like that in Figure 9 in the first signature to omit the approver signature from the content signed by the first signer. This allows the approver to affix the signature without invalidating the signature of the first signer. More generally, a digital signature should subtract from the signed content the portion of the document that is expected to change during future steps of the business process workflow.

8. AN OFFICE DOCUMENT MASHUP

This section reports on an experimental prototype designed to illustrate the feasibility and advantages of con-

necting interactive office documents to the business process. The prototype was dubbed 'Dual Forms' because it exploits the XForms markup support in two rich document formats, XFDL [6] and ODF [15].

At the document level, the XFDL form has a file attachment/containment capability, and it also has a full document submission capability. So, the XFDL form is used as the container and transport envelope for an ODF office document representing a complex contract or agreement. The XFDL form also has digital signature support, and once affixed, a digital signature protects not only the XFDL form, but also the ODF attachment within it.

The ODF document provides editable free-flowing text for the complex, multi-page contract. It may also including rich content elements like pie charts and bar graphs to serve as visual aids. The XFDL form provides a wizard-like front-end for the contract to help the user enter data systematically. This interaction may include accessing to SOAP-based web services and ATOM services from the XFDL form.

The client-side usage pattern for a 'Dual Form' begins with receiving the XFDL form, from a web resource or an email. Figure 10 shows a screen for a hypothetical consumer loan application in which end-users receive a step-by-step wizard experience to help guide them through the process of providing the more structured data required by the overall office document application.

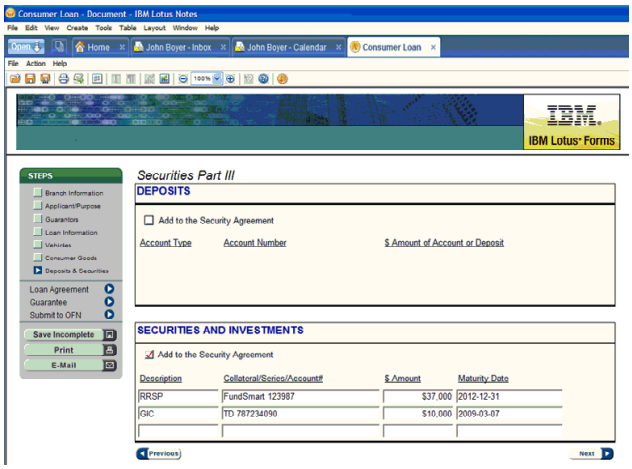
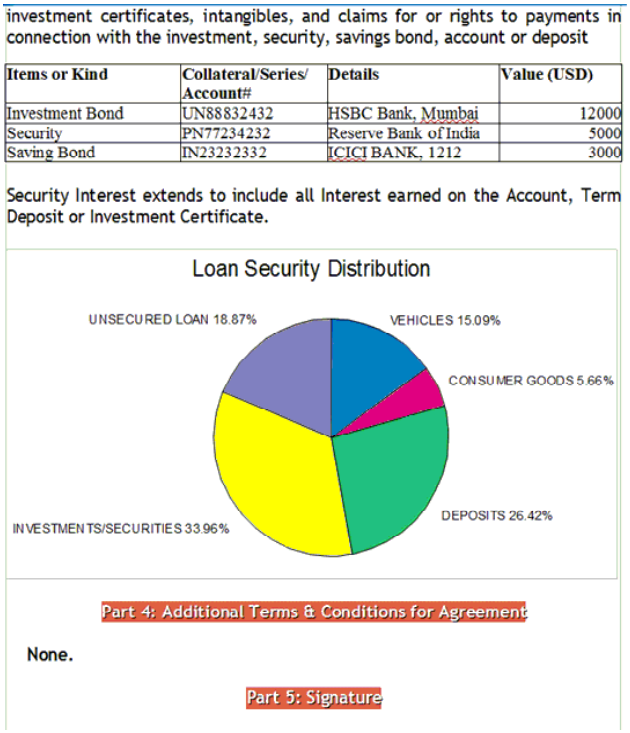


Figure 10: The first phase of a Dual Form is the step-by-step wizard experience for a hypothetical consumer loan application. This could just as easily be a patent filing, a healthcare or insurance document, or any kind of contract.

The author of the Dual Form application decides at what point the end-user is allowed to proceed to the ODF document embedded in the XFDL form. Figure 11 depicts some of the sample content for the hypothetical consumer loan application, which can include the structured data along with rich content items that serve as visual aids as well as free-flowing text for customizing the agreement with specialized terms and conditions.

When the end-user finishes interacting with the office document, a toolbar button allows the end-user to return to the XFDL form view. Typically, the same condition that switches the Dual Form to the office document view will also advance the wizard view to the signatures step, if the


investment certificates, intangibles, and claims for or rights to payments in connection with the investment, security, savings bond, account or deposit

Items or Kind	Collateral/Series/ Account#	Details	Value (USD)
Investment Bond	UN88832432	HSBC Bank, Mumbai	12000
Security	PN77234232	Reserve Bank of India	5000
Saving Bond	IN23232332	ICICI BANK, 1212	3000

Security Interest extends to include all Interest earned on the Account, Term Deposit or Investment Certificate.

Part 4: Additional Terms & Conditions for Agreement

None.

Part 5: Signature

Figure 11: The second phase of a Dual Form is the office document experience for a hypothetical consumer loan application. This may include some structured data such as names, addresses and amounts, some rich content items such as pie charts and bar graphs, and some free-flowing unstructured data such as special terms and conditions.

application includes digital signatures, or to the document submission step. Figure 12 depicts the completion of a digital signature over the XFDL form, including the ODF document it contains.

The final step of interacting with the Dual Form is to press a button to submit it to the server-side business process. The document could enter the next stage of a workflow, such as an approval, or it could represent a completed agreement that kicks off a business transaction and is otherwise saved to a content repository for future reference.

From a technical perspective, the above usage pattern was supported by creating an an office document mashup of the Lotus Forms viewer and the Lotus Symphony ODF editors under the Lotus Notes/Expeditor Composite Application Framework. Figure 13 illustrates the block wiring diagram for Dual Forms mashup.

Based on an event in the XFDL form that can be controlled by the form author, the mashup switches from the form view to the ODF view. When this occurs, the view component uses the Lotus Form Viewer's javascript API to obtain the XForms instance data and the ODF attachment and set them into the matching properties of form view component. This triggers the composite application mashup wires to invoke the corresponding setter methods in the ODF view component. The setter method implementations use the UNO API to instantiate the ODF content and update it with the XML data. Thus, the ODF content is

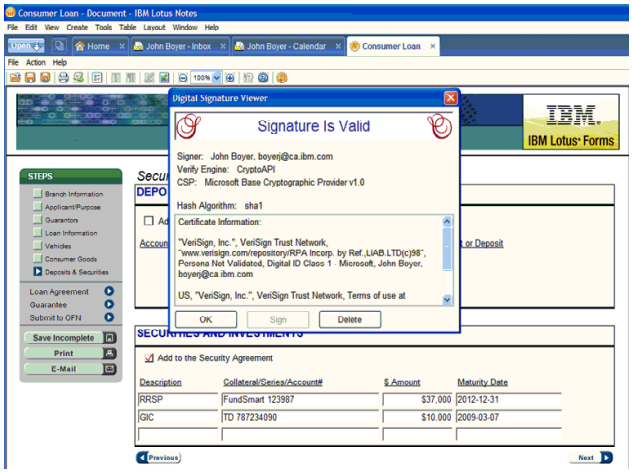


Figure 12: In the completion phase of the Dual Form for the hypothetical consumer loan application, the end-user affixes a digital signature over the full document, including the ODF content, and then submits the Dual Form containing the ODF and the signature to the server.

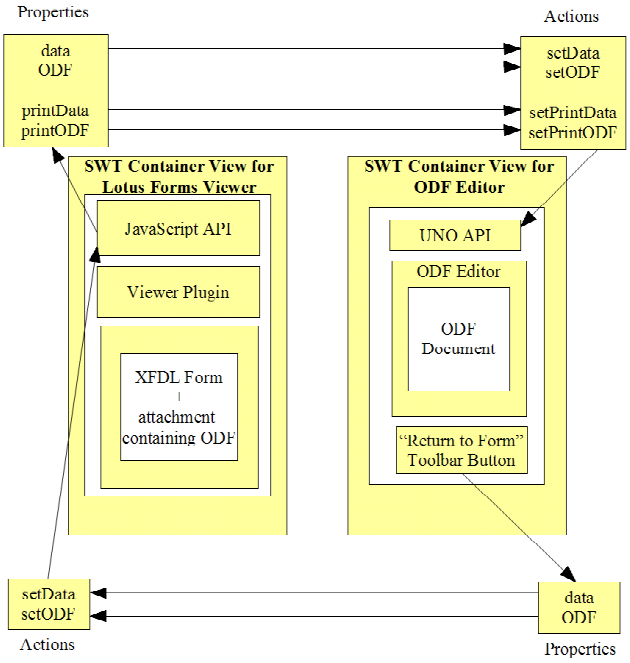


Figure 13: The block wiring diagram for the Dual Forms office document mashup.

only rendered after it is updated by the latest XML data entered into the XFDL form.

The end-user may edit the XML data using ODF form controls. The end-user may also edit the free-flowing text of the ODF, for example to add special terms and conditions beyond those that might reasonably appear in a document template. When the user hits a 'return to form' button on the toolbar of the ODF view component, the UNO API is again used to obtain the XML data from the XForms instance as well as a serialization of the ODF content. These are used to set the data and ODF properties of the ODF view component. The mashup wires then invoke the corresponding setter methods in the XFDL form view. The implementations of the setter functions use the javascript API to push the updated ODF content and the XML data into the running XFDL form. Thus, when the end-user affixes a digital signature onto the XFDL form, the form contains the latest ODF content and is rendering the correct data as amended during the ODF view experience.

The mashup wiring for the print capability is the same as the Form to ODF view wiring described above. The main difference is that ODF view component simply prints the ODF without taking the focus from the form view. For both view and print operations, the simplicity of the mashup wiring is made possible by the fact that both rich document formats have an underlying basis in XForms.

9. CONCLUSION AND FUTURE WORK

This paper first recognizes that the evolution of Web 2.0 is following the same trend as the personal computer revolution– a trend from *content democratization* (e.g. word processing, the wiki) to *process democratization* (e.g. the spreadsheet, declarative markup for the Web 2.0 client tier). This paper then provides a vision of connecting office documents to the business process based on having intelligent and interactive, secure, mobile ODF documents. One or many collaborators are able to create, modify and secure content based on an ODF document that freely transitions between the client-side and server-side as needed.

An office document mashup called 'Dual Forms' was presented to illustrate the feasibility and advantages of imbuing office documents with rich interactivity, web service access, digital signature security, and full document mobility within business processes. This prototype combined the Lotus Symphony ODF editors with the document-centric precision electronic forms of Lotus Forms under the Lotus Notes/Expeditor Composite Application Framework.

Fully realizing this vision depends mainly on strengthening the integration between ODF and a companion W3C standard that ODF already contains. XForms became a W3C standard in 2003, and XForms 1.1 reached the candidate standard in November 2007. XForms 1.1 allows the full realization of the vision presented here due to enablement of features that allow it to be used with a service-oriented architecture (SOA). If the SOA-enabled Web 2.0 is the medium, then the XForms-based ODF document is the message. In the philosophy of Marshall McLuhan, they are one.

10. ACKNOWLEDGMENTS

The author gratefully acknowledges the Biztech Extreme Blue grant funding from the IBM China Software Development Lab as well as the diligence of the implementation team for the "Dual Forms" office document mashup, including Eric Dunn, Maureen Kraft, Jun Liu, Mihir Shah, He Feng Su, and Saurabh Tiwari.

11. REFERENCES

[1] Andrea R. de Andrade, Ethan V. Munson, and Maria da G. Pimentel. Document-based Approach to the Generation of Web Applications. *Proceedings of the 2004 ACM Symposium on Document Engineering*, pp. 45-47, Oct. 28-30, 2004. Milwaukee, Wisconsin, USA.

[2] Mark Birbeck and John M. Boyer (eds.). *The Ubiquity XForms Processor*. http://code.google.com/p/ubiquity-xforms/

[3] Barclay Blair and John M. Boyer. XFDL: Creating Electronic Commerce Transaction Records Using XML. *Proceedings of the* 8^{th} *World Wide Web Conference* and *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 31, pp. 1611-1622, 1999.

[4] Rebecca Blood. Weblogs: A history and perspective. *Weblog: rebecca's pocket*. September 7, 2000. Available at: http://www.rebeccablood.net/essays/weblog_history.html

[5] John M. Boyer. Bulletproof Business Process Automation: Securing XML Forms with Document Subset Signatures. *Proceedings of the ACM Workshop on XML Security*, October 31, 2003. Fairfax, VA, USA.

[6] John M. Boyer. Enterprise-level Web Form Applications with XForms and XFDL. *Proceedings of the XML 2005 Conference and Exposition*, November 14-18, 2005. Atlanta, GA, USA. Available at: http://www.idealliance.org/proceedings /xml05/ship/74/XFormsAndXFDL_Boyer.HTML

[7] John M. Boyer. Applying XML Signatures to XForms Documents. *Proceedings of XML 2006 Conference and Exposition*, December 5-7, 2006. Boston, MA, USA. Available at: http://2006.xmlconference.org /proceedings/100/slides.pdf

[8] John M. Boyer (ed.). *XForms 1.1*. W3C Candidate Recommendation, November 29, 2007. Available at: http://www.w3.org/TR/2007/CR-xforms11-20071129/

[9] John M. Boyer and Mikko Honkala. The XForms Computation Engine: Rationale, Theory and Implementation Experience. *Proceedings of the* 6^{th} *IASTED International Conference on Internet and Multimedia Systems and Applications*, pp. 196-204, August 12-14, 2002. Kauai, Hawaii, USA.

[10] John M. Boyer, Tim Bray and Maureen Gordon (eds). *Extensible Forms Description Language (XFDL) 4.0*. W3C Note. September 2, 1998. Available at: http://www.w3.org/TR/NOTE-XFDL

[11] John M. Boyer, Merlin Hughes and Joseph Reagle (eds). *XML-Signature XPath Filter 2.0*. W3C Recommendation. November 8, 2002. Available at: http://www.w3.org/TR/xmldsig-filter2/

[12] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau (eds). *Extensible Markup Language (XML) 1.0 (Fourth Edition)* W3C Recommendation, September 29, 2006. Available at: http://www.w3.org/TR/2006/REC-xml-20060816/

[13] Richard Cardone, Danny Soroker, and Alpana Tiwari. Using XForms to Simplify Web Programming. *Proceedings of the* 14^{th} *World Wide Web Conference*, pp. 215-224, May 10-14, 2005. Chiba, Japan.

[14] W. Cunningham and B. Leuf. *The Wiki Way*, New York, Addison-Wesley, 2001.

[15] Patrick Durusau, Michael Brauer, and Lars Opperman (eds.). *Open Document Format for Office Applications (OpenDocument) v1.1*. OASIS Standard, Feb. 1, 2007. Available at: http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.odt

[16] Donald Eastlake, Joseph Reagle, David Solo, Mark Bartel, John M. Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. *XML-Signature Syntax and Processing*. W3C Recommendation, February 12, 2002. Available at: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

[17] Becky Gibson. Enabling an Accessible Web 2.0. *W4A2007 - Keynote*, Co-located with the 16^{th} International World Wide Web Conference, May 7-8, 2007. Banff, Canada.

[18] Robert J. Glushko and Tim McGrath. Document Engineering for e-Business. *Proceedings of the 2002 ACM Symposium on Document Engineering*, pp. 42-48, Nov. 8-9, 2002. McLean, Virginia, USA.

[19] Joe Gregorio and Bill de hOra (eds.). *The Atom Publishing Protocol*. IETF RFC 5023, October 2007. Available at: http://www.rfc-editor.org/rfc/rfc5023.txt

[20] Charles Hill, Robert Yates, Carol Jones, Sandra L. Kogan. Beyond predictable workflows: Enhancing productivity in artful business processes. *IBM Systems Journal* vol. 45, no. 4, pp. 663-682, 2006.

[21] Mikko Honkala and Mikko Pohja. Multimodal interaction with XForms. *Proceedings of the* 6^{th} *International Conference on Web Engineering*, pp. 201-208, July 11-14, 2006. Palo Alto, California, USA.

[22] Angelo Di Iorio and Fabio Vitali. From the Writable Web to Global Editability. *Proceedings of ACM Hypertext 2005*, pp. 35-45, Sept. 6-9, 2005. Salzburg, Austria.

[23] Y. S. Kuo, N.C. Shih, Lendle Tseng, and Hsun-Cheng Hu. Generating Form-Based User Interfaces for XML Vocabularies. *Proceedings of the 2005 ACM Symposium on Document Engineering*, pp. 58-60, Nov. 2-4, 2005. Bristol, UK.

[24] Y. S. Kuo, Lendle Tseng, Hsun-Cheng Hu, and N.C. Shih. An XML Interaction Service for Workflow Applications. *Proceedings of the 2006 ACM Symposium on Document Engineering*, pp. 53-55, Oct. 10-13, 2006. Amsterdam, The Netherlands.

[25] Dongxi Liu, Zhenjiang Hu, Masato Takeichi. An Environment for Maintaining Computation Dependency in XML Documents. *Proceedings of the 2005 ACM Symposium on Document Engineering*, pp. 42-51, Nov. 2-4, 2005. Bristol, UK.

[26] Tom Malloy. The Future of Documents. *Proceedings of the 2005 ACM Symposium on Document Engineering*, Keynote abstract, Nov. 2-4, 2005. Bristol, UK.

[27] Vincent Quint and Irene Vatton. Toward Active Web Clients. *Proceedings of the 2005 ACM Symposium on Document Engineering*, pp. 168-176, Nov. 2-4, 2005. Bristol, UK.

[28] Simon Thompson, Peter R. King, and Patrick Schmitz. Declarative Extensions of XML Languages. *Proceedings of the 2007 ACM Symposium on Document Engineering*, pp. 89-91, Aug. 28-31, 2007. Winnipeg, Canada.