A
Dissertation
On

REDESIGN OF XFORMS + XFDL AND ENRICHING WITH FEATURES FROM XFA

Submitted in Partial fulfillment of the requirements
for the award of Degree of

MASTER OF ENGINEERING
(Computer Technology and Application**)**

Submitted By

KALPESH KUMAR MEENA
College Roll No: 12/CTA/05
University Roll No. 2011

Under the Guidance of:
Prof. D Roy Choudhury
Department Of Computer Engineering
Delhi College of Engineering, Delhi

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
DELHI UNIVERSITY
2005-2007

CERTIFICATE

This is to certify that the work contained in this dissertation entitled **"Redesign of XForms + XFDL and Enriching with features from XFA"** by **Kalpesh Kumar Meena** in the requirement for the partial fulfillment for the award of the degree of **Master of Engineering** in Computer Technology & Application, Delhi College of Engineering is an account of his work carried out under my guidance in the academic year 2006-2007.

This work embodies in this dissertation has not been submitted for the award of any other degree to the best of my knowledge.

Prof. D Roy Choudhury
Head of Department
Department of Computer Engineering
Delhi College of Engineering
Delhi

Dr. S C Gupta
Sr. Technical Director
National Informatics Center
Delhi

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extent my heartiest felt gratitude to everybody who helped me throughout the course of this project.

It is distinct pleasure to express my deep sense of gratitude and indebtedness to my learned supervisors Dr. S C Gupta and Prof. D Roy Choudhury for their invaluable guidance, encouragement and patient reviews. Their continuous inspiration only has made me complete this dissertation. Both of them kept on boosting me time and again for putting an extra ounce of effort to realize this work.

I would like to specially thank Minakshi Anand and Dhiraj Kumar Singh for their constant support in the lab. I would also like to take this opportunity to present my sincere regards to my teachers Prof. Goldie Gabrani, Dr. S. K. Saxena, Mrs. Rajni Jindal, Mr. Manoj Sethi and Mr. Rajeev Kumar for their support and encouragement.

I am grateful to my parents for their moral support all the time, they have been always around to cheer me up in the odd times of this work. I am also thankful to my classmates for their unconditional support and motivation during this work. Living at DCE with them has been a lifetime experience for me, all the time we spend together enjoying life to its fullest, the birthday parties, placement parties and photo sessions discussing new topic or technology would remain with me forever.

I want to thank the IBM Research Scientist Dr. John M. Boyer for his valuable suggestions which helped us a lot during this project. Last but not least, special thanks to the members of World Wide Web Consortium (W3C).

Kalpesh Kumar Meena
M.E. (Computer Technology and Applications)
College Roll No. 12/CTA/05
University Roll No. 2011
Department of Computer Engineering
Delhi College of Engineering, Delhi-110042

ABSTRACT

At the time HTML was developed, it was identified that the simplest way to create web pages, was to use declarative tags. Initially, web was only perceived as the data distribution medium and not for data collection. But as web pages became more and more interactive, HTML was no longer sufficient. Today, we find form code containing a large amount of scripting. To simplify coding of such forms, W3C introduced a declarative alternative to scripting called XForms (An XML based language).

XForms is a device-independent standard and requires a specific presentation language like XFDL, XFA for display of information. XFDL (eXtensible Forms Description Language) has graduated to include XForms whereas XFA (XML Forms Architecture) still follows the approach without XForms. Though XFA does not include XForms, it has certain important features like the prototype tag which are missing in XFDL. But XFDL provides for more security, important for e-commerce transactions. After a detailed comparison between the two, we found XFDL to be a better alternative.

So, **in this thesis, we have included the missing features into XFDL. Use attribute has been included in XFDL to provide the same functionality as the prototype tag of XFA. Also, some of XFA functions are also included. Also, we have simplified some of XFDL's existing tags. As it integrates XForms, we have simplified the XForms tags as well. This helps make XFDL, a more complete and designer-friendly language.**

A converter has been designed which converts the simplified XFDL code to the original code. This converter is designed using DOM package within JAXP (Java API for XML Processing).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Program Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
DTD	Document Type Definition
HTML	Hyper Text Markup Language
JAXP	Java API for XML Parsing
MIP	Model Item Properties
MVC	Model-View-Controller
P3P	Platform for Privacy Preferences
RFC	Request for Comments
SAX	Simple API for XML
UI	User Interface
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XHTML	eXtensible Hyper Text Markup Language
XFDL	eXtensible Forms Description Language
XML	eXtensible Markup Language
XSL	eXtensible Style Sheets Language

1. INTRODUCTION

1.1 Evolution of Forms Technology

As a general rule, the more interactive a web site is, the more heavily the site's designers rely on web forms, a general term for all different kinds of technologies used to gather information from users. Without forms, web sites are far less interesting. Form-less web sites were the norm in the early days of the Web and provided a one-way deluge of static information.

The addition of forms to Hypertext Markup Language (HTML), the primary language used in web pages, launched an entirely new way of surfing the Web. Using HTML forms, searching for information became possible on a worldwide scale. Sites such as Yahoo! quickly became the most popular "portals" of entry on the Web. Later, as developers pushed the limits of forms technology farther, web sites became even more interactive and customizable.

Shortly after the initial tempering of HTML, various individuals began considering the usefulness of forms alongside hypertext. HTML Version 2.0, as presented in a document called Request for Comments (RFC) 1866, was the first time that web forms were seriously considered for standardization. That RFC captured HTML as found in common use prior to June 1994. At this point, HTML already included forms, thanks to a 1993 proposal called HTML+.

Care and maintenance of the HTML family of specifications have since been handed over to the World Wide Web Consortium, or W3C. The last non-XML-based version of HTML was version 4.01, which didn't change forms processing much. New development of the standard is taking place on a closely related technology called XHTML, where the X indicates an XML foundation. XHTML 1.0 and 1.1 were largely concerned with details of the transition to XML and ways to combine vocabularies, not with major changes to the language.

XHTML 2.0, in contrast, is making some improvements that aren't compatible with earlier flavors of HTML. The largest such change is the adoption of XForms as a replacement for the older HTML forms technology.

The XForms standard is device- independent and therefore can be combined with any presentation language like HTML, XFDL(eXtensible Forms Description Language), SMIL(Simple Multimedia Integration Language), SVG(Scalar Vector Graphics) etc.

1.2 MOTIVATION

I always faced a lot of problems coding forms in HTML. Though, at a time when HTML was developed, it changed the entire web scenerio, coding of today's forms in HTML requires use of large amount of scripting and finally the code becomes unmanagable. So, much simpler standards based on XML are being developed. One such standard, I happened to use was XForms. It follows the basic rule that everything can be specified using markup, and there is no need of scripting. But, XForms has certain tags that difficult are to understand. So,I have tried to simplify these.

I found 2 new generation languages – XFDL and XFA as suitable presentation options for XForms. I prefer the first as it already incooprates XForms and provides more security. But, I identified some XFA features missing in XFDL and added those to it.

1.3 HTML

HTML is a non-proprietary format based upon SGML, and can be created and processed by a wide range of tools, from simple plain text editors - you type it in from scratch- to sophisticated WYSIWYG authoring tools. HTML uses tags such as `<h1>` and `</h1>` to structure text into headings, paragraphs, lists, hypertext links etc.

The introduction of the forms chapter in HTML 4.01 reads: "An HTML form is a section of a document containing normal content, markup, special elements called controls (checkboxes, radio buttons, menus, etc.), and labels on those controls. Users generally 'complete' a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing (e.g., to a web server, to a mail server, etc.)."

Forms represent a structured exchange of data. In HTML forms, the structure of the collected data, called a *form data set*, is a set of name/value pairs. The names and values that are included in this set are solely determined by the controls present within the form, so that adding a new control element, as well as adding to the user interface, also adds a new name/value pair to the data set. Many authors take for granted this basic violation of the separation between the data layer and the user interface layer—a problem that XForms has gone to considerable lengths to alleviate.

1.4 XML

XML is a text-based markup language that is fast becoming the standard for data interchange on the Web. XML(e**X**tensible **M**arkup **L**anguage) provides a tag-based syntax for structuring data and applying markups to documents. But unlike HTML, XML tags *identify* the data, rather than specifying how to display it. Where an HTML tag says something like "display this data in bold font"(<b>...</b>), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example: <message>...</message>).
There are a number of reasons for XML's surging acceptance. This section lists a few of the most prominent.

1. Plain Text
Since XML is not a binary format, you can create and edit files with anything from a standard text editor to a visual development environment. That makes it easy to debug your programs, and makes it useful for storing small amounts of data. At the other end of the spectrum, an XML front end to a database makes it possible to efficiently store large amounts of XML data as well. So XML provides **scalability** for anything from small configuration files to a company-wide data repository.

2. Data Identification
XML tells you what kind of data you have, not how to display it. Because the markup tags identify the information and break up the data into parts, an email program can process it, a search program can look for messages sent to particular

people, and an address book can extract the address information from the rest of the message. In short, because the different parts of the information have been identified, they can be used in different ways by different applications.

3. Easily Processed

As mentioned earlier, regular and consistent notation makes it easier to build a program to process XML data. For example, in HTML a <dt> tag can be delimited by </dt>, another <dt>, <dd>, or </dl>. That makes for some difficult programming. But in XML, the <dt> tag must always have a </dt> terminator, or else it will be defined as a <dt/> tag. (Otherwise, the XML parser won't be able to read the data.) And since XML is a **vendor-neutral standard,** you can choose among **several XML parsers,** any one of which takes the work out of processing XML data.

4. Hierarchical

Finally, XML documents benefit from their hierarchical structure. Hierarchical document structures are, in general**, faster to access** because you can drill down to the part you need, like stepping through a table of contents.

1.5 OBJECTIVE

The objective of this thesis is to

- Introducing modifications to XForms + XFDL in order to simplify coding using these languages.
- Design a converter that converts the simplified code into XForms + XFDL code which can be displayed using the IBM Workplace Forms Viewer.
- Compare the XFDL and XFA Specifications and introduce missing features into XFDL.

2. XFORMS

2.1 Introduction

XForms has been designed on the basis of several years' experience with HTML forms. HTML Forms have formed the backbone of the e-commerce revolution, and having shown their worth, have also indicated numerous ways they could be improved.

The primary difference when comparing XForms with HTML Forms, apart from XForms being in XML, is the **separation of the data** being collected **from the markup** of the controls collecting the individual values. By doing this, it not only makes XForms more tractable by making it clear what is being submitted where, it also eases reuse of forms, since the underlying essential part of a Form is no longer irretrievably bound to the page it is used in.

A second major difference is that XForms, while designed to be integrated into XHTML, is no longer restricted only to be a part of that language, but may be integrated into any suitable markup language. XForms has striven to improve authoring, reuse, internationalization, accessibility, usability, and device independence. Here is a summary of the primary benefits of using XForms:

1. **Strong typing -** Submitted data is strongly typed and can be checked using off-the-shelf tools. This speeds up form filling since it reduces the need for round trips to the server for validation.

2. **XML submission** - This obviates the need for custom server-side logic to marshal the submitted data to the application back-end. The received XML instance document can be directly validated and processed by the application back-end.

3. **Existing schema re-use** - This obviates duplication, and ensures that updating the validation rules as a result of a change in the underlying business logic does not require re-authoring validation constraints within the XForms application.

4. **External schema augmentation** - This enables the XForms author to go beyond the basic set of constraints available from the back-end. Providing such additional constraints as part of the XForms Model enhances the overall usability of the resulting Web application.

5. **Internationalization** - Using XML 1.0 for instance data ensures that the submitted data is internationalization ready.

6. **Enhanced accessibility** - XForms separates content and presentation. User interface controls encapsulate all relevant metadata such as labels, thereby enhancing accessibility of the application when using different modalities. XForms user interface controls are generic and suited for device-independence.

7. **Multiple device support** - The high-level nature of the user interface controls, and the consequent intent-based authoring of the user interface makes it possible to re-target the user interaction to different devices.

8. **Less use of scripting** - By defining XML-based declarative event handlers that cover common use cases, the majority of XForms documents can be statically analyzed, reducing the need for imperative scripts for event handlers.

XForms has **three parts—XForms model, instance data, and user interface**—it separates presentation from content, allows reuse, gives strong typing—reducing the number of round-trips to the server, as well as offering device independence and a reduced need for scripting.

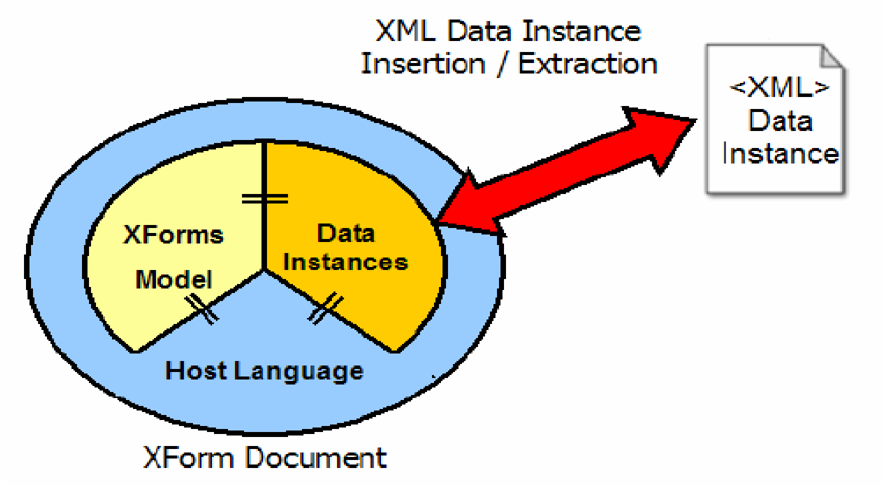


Fig. 2.1 Architecture of XForms

An important concept in XForms is that forms collect data, which is expressed as XML **instance data**. Among other duties, the XForms Model describes the structure of the instance data. This is important, since like XML, forms represent a structured interchange of data. Workflow, auto-fill, and pre-fill form applications are supported through the use of instance data.

2.2 Basis of XForms

XForms is based on various XML Technologies – XML Schema, XML Events, XPath, Namespaces etc. As can be seen from Fig.2.2, various standards have been developed based on XML and XForms is in turn based on these. Thus, XForms occurs at the third level in the XML hierarchy.

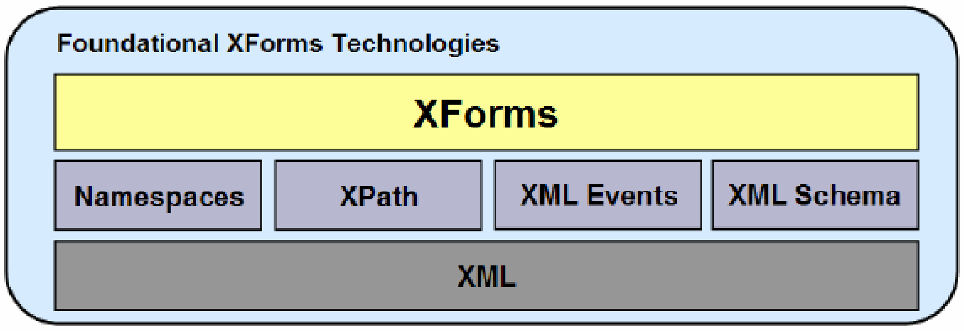


Fig. 2.2 Basis of XForms

2.2.1 XML Namespaces

Because XML allows designers to chose their own tagnames, it is possible that two or more designers may choose the same tagnames for some or all of their elements. XML namespaces provide a way to distinguish deterministically between XML elements that have the same local name but are, in fact, from different vocabularies. This is done by **associating an element with a namespace**. A namespace acts as a scope for all elements associated with it. Namespaces themselves also have names. A namespace name is a uniform resource identifier (URI). Such a URI serves as a unique string and need not be able to be dereferenced. The namespace name and the local name of the element together form a globally unique name known as a ***qualified name***.

Namespace declarations appear inside an element start tag and are used to map a namespace name to another, typically shorter, string known as a ***namespace prefix***. The syntax for a namespace declaration is **xmlns:prefix='URI'**. It is also possible to map a namespace name to no prefix using a default namespace declaration. The syntax for a **default namespace declaration** is **xmlns='URI'**. In both cases, the URI may appear in single quotes (') or double quotes ("). Only one default namespace declaration may appear on an element. Any number of nondefault namespace declarations may appear on an element, provided they all have different prefix parts.

2.2.2 XML Schema

An XML schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).They are XML-based alternative to DTD(Document Type Definition). An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

Advantages of XML Schema over the DTD are :

1. XML Schemas are richer and **more powerful than DTDs**
2. XML Schemas **support data types**
3. XML Schemas **use XML Syntax** - Some benefits of that XML Schemas are written in XML are – we can use XML editor to edit Schema files, XML parser to parse Schema files, manipulate Schema with the XML DOM and transform with XSLT.
4. XML Schemas **are Extensible**

Well formed and Valid XML

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.With XML Schemas, most of these errors can be caught by your validating software. An XML file that conforms to a Schema is called a valid XML file.

2.2.3 XPath

XPath is a language for addressing parts of an XML document. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax. XPath gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document. **XPath models an XML document as a tree of nodes**. There are different types of nodes, including element nodes, attribute nodes and text nodes.

Simple XPath expressions resemble file system paths, except that instead of navigating across directories and files, XPath expressions navigate across XML *nodes.* For example, the expression:

/html/head/title

represents an absolute path through XML, starting at a special root node, then progressing through child elements `html`, `head`, and `title`. The XML referenced by this path might look something like this:

```
<html>
```

```
  <head>
    <title>Push Button Paradise</title>
...
```

Since XML names can be qualified with a namespace, it's also possible to use colonized names at any step. Relative paths are also possible, in which case it's important to know what the *context node* (similar in concept to the current directory) is. Additionally, **attributes can be addressed with a leading `@` character**, leading to XPath expressions like this:

```
html:head/xforms:model/@id
```

Note that when the leading slash is omitted, the path expression is relative.

Path expressions can be said to return a *node-set*. Both of the above examples conveniently returned a node-set consisting of a single node, but in the general case, node-sets can have zero, one, or a multitude of nodes. XForms includes a *first node rule*, that in certain circumstances, will reduce a larger node-set down to a single node, namely, the first one according to the order the elements appear in the document. Also, node-sets can be filtered manually using a ***predicate*.** Predicates are identified using square brackets as follows:

```
purchaseOrder/items/item[3]
```

XPath expressions can also be more than just paths, and can be thought of as a kind of **lightweight scripting language**. Besides node-sets, an expression can evaluate to a Boolean value, a string, or a number. For example, the expression:

```
string-length('hello world')
```

would always return 11 as a number, and the expression:

```
purchaseOrder/subtotal * instance('taxtable')/tax
```

represents a full-blown calculation that might appear in a real-world form. On the right-hand side of the multiplication symbol, note that the path expression begins with a function call that can return a node-set from another location (a different XForms instance, in this case).

Context

Just as a relative directory path is relative to a current directory, XPath expressions are evaluated relative to a "context," which consists of the following:

- A context node
- A pair of non-zero positive integers (the context position and the context size)
- Variable bindings (not used in XForms)
- A function library
- The set of namespace declarations in scope for the expression

In Xpath, default namespace is *not* part of the context. Therefore, every namespace referenced in an XPath expression needs to be associated with a specific prefix at the point where the XPath expression occurs.

An XPath expression of '`.`' selects the context node, and '`..`' selects the parent node of the context node. Any expression that begins with '`/`' is an absolute path and independent from the context node.

2.2.4 XML Events

An ***event*** is the representation of some asynchronous occurrence (such as a mouse click on the presentation of the element, or an arithmetical error in the value of an attribute of the element, or any of unthinkably many other possibilities) that gets associated with an element (*targeted* at it) in an XML document.

In the DOM model of events, the general behavior is that when an event occurs it is *dispatched* by passing it down the document tree in a phase called ***capture*** to the element where the event occurred (called its *target*), where it then may be passed back up the tree again in the phase called ***bubbling***. In general an event can be responded to at any element in the path (an *observer*) in either phase by causing an action, and/or by stopping the event, and/or by cancelling the default action for the event.

2.3 The XForms Model

XForms Model is the name given to the form description. That name was chosen mainly because it wasn't "data model," but also to evoke thoughts of the **Model-View-Controller (MVC) design pattern** in programming. In MVC, a model contains all the essential data, and one or more views provide a viewpoint to examine or interact with the data. The XForms Model is analogous to a MVC model, and form controls serve the function of views. (There's nothing that directly maps to a controller in XForms, though portions of the processing model and XForms Events play a similar role.)

A ***model item*** is the name for an XPath node with the addition of certain XForms properties, formally called *model item properties*. The connection between model item properties and form controls is called *binding*, which is accomplished through a set of XML elements that comprise the XForms Model.

2.3.1 Structural Elements

The XForms Model is made up of a number of different elements, outlined here.

1. **The model Element -** This element is the local root of the definition of the XForms Model. It is typically found in a non-rendered area of the containing document. **Eg -**the `head` section in XHTML can contain an XForms Model.

2. **The instance Element -** This element serves as a container for initial instance data. The contents of this element are simply data that will be both read and written during form interaction, nothing more.
   Instead of inline content, `instance` may use Linking Attributes (that is to say, `src`) to point to external instance data.

3. **The bind Element -** This element establishes conditions that are continuously applied to the instance data. With instance data defined neatly by the `instance` element, the question remains of how to annotate instance data nodes with properties necessary for forms. Each model item property is represented by an attribute on this element- `type`**,** `readonly`**,** `Required`, `relevant`, `calculate`, `constraint`, `p3ptype`.
   The properties are applied through an additional attribute, `nodeset`, which selects a node-set.

4. **The submission Element** – Specifies how,where and what data is submitted.

2.3.2 Binding Attributes

The Common attribute collection contains the Binding Attributes—Single Node and Node-set. A number of situations in XForms call for a reference into instance data. *Binding attributes* provide this feature. The following section describes these attributes :

- **ref** - Whenever the intent of the binding attributes is to select a single node, the `ref` attribute will be present. It contains an XPath path expression. In cases where the selected node-set happens to have more than one node, the *first node rule* applies, which removes all nodes other than the first, according to the order the nodes appear in the document.

- **nodeset** - Whenever the intent of the binding attributes is to select a node-set of any size, the `nodeset` attribute will be present. It contains an XPath path expression.

- **model** - In larger or more complex documents, it will be common to have multiple XForms Models. When this is the case, an additional attribute is needed to indicate to which XForms Model the binding attaches. The value of this attribute is of type IDREF, and so a `model` element in the same document must have an attribute of type ID with a matching value.

- **bind** - In some cases, such as when a graphic design professional who isn't concerned with XPath is laying out a form, it isn't desirable to have XPath strewn about on every set of binding attributes. The `bind` attribute, which takes precedence over any of `ref`, `nodeset`, or `model`, refers back to an already-defined node-set on a `bind` element. The value of this attribute is of type IDREF, and so a `bind` element in the same document must have an attribute of type ID with a matching value.

It's worth noting that the term binding, as used in XForms, can refer to two separate things. ***UI Binding*** occurs on an attribute of a form control element, and binds the form control to a particular model item. In dynamic forms, the association to a model item can jump around, causing the form control to be a window to different parts of the data at different times. The other use of binding, ***Model Binding***, occurs on the element `bind`, selecting an entire node-set to which a set of model item properties gets applied. It is a serious problem to have a dynamic model binding expression, since that complicates life behind-the-scenes for an XForms Processor, which can cause difficult-to-detect errors.

2.3.3 Binding

A bind has two ends, one side in the *XForms Model*, and the other side at a *form control*. On the `bind` element within the XForms Model, the `nodeset` attribute holds the **Model Binding Expression**. On the other end, in the user interface, is the **UI Binding Expression.** This end may be bound two ways, using either IDREFs or XPath.

1. **With IDREFs** - The recommended way to perform binding is to put an `id` attribute on each `bind` element, and refer back to this with a `bind` attribute on each form control:

```
<!-- in the XForms Model -->
   <xforms:bind nodeset="email" id="mybind"
   required="true()"/>
       ...
<!-- later in the document -->
   <xforms:input bind="mybind"...>
```

This approach is distinguished by the use of the **`bind` attribute on form controls**. The main advantage of this approach is that it maintains separation between the model and the view. If the structure of the instance data were to change, only the attributes on the `bind` elements would need to be updated.

2. **With XPath** - Another way to bind is with XPath expressions on the form controls:

```
<!-- in the XForms Model -->
   <xforms:bind nodeset="email" id="mybind"
   required="true(  )"/>
       ...
<!-- later in the document -->
   <xforms:input ref="email"...>
```

This approach is distinguished by the use of **`ref` attributes** on form controls. Many view this approach as simpler, since it cuts out one level of indirection. It is also more fragile, however, since the XPath expressions to locate nodes appear in two places. If the structure of the instance data were to change, both the attributes on the `bind` element and the `ref` attributes on the form controls would need to change.

2.4 XForms User Interface

The following table lists the various form controls with their specific purposes –

Purpose	Form control
Entering information into the form	input
Writeonly control (non-readable,used for passwords)	secret
Displaying information	output
Upload a file(eg- image) to a form	upload
To get a bounded value	range
To trigger an action	trigger
Submission of data	submit
Selection of multiple values from a set	select
Selection of one value from a set	select1
Dynamic Presentation – To dynamically vary the number of rows in the table	repeat
Dynamically choose from one of the several options for display	switch

Table 2.1 XForms Form Controls

For detailed code, refer to section 3.3 (XForms and XFDL).

2.5 Xorms Actions

An ***event*** is the representation of some asynchronous occurrence (such as a mouse click on the presentation of the element, or an arithmetical error in the value of an attribute of the element, or any of unthinkably many other possibilities) that gets associated with an element (*targeted* at it) in an XML document.

In the DOM model of events, the general behavior is that when an event occurs it is *dispatched* by passing it down the document tree in a phase called ***capture*** to the element where the event occurred (called its *target*), where it then may be passed back up the tree again in the phase called ***bubbling***. In general an event can be responded to at any element in the path (an *observer*) in either phase by causing an action, and/or by stopping the event, and/or by cancelling the default action for the event.

An ***action*** is some way of responding to an event; a ***handler*** is some specification for such an action, for instance using scripting or some other method. A ***listener*** is a binding of such a handler to an event targeting some element in a document.

2.5.1 The Old Way

In the design of HTML forms, script is used whenever some specific action is needed. For example, a form might have a button that copies values from a "ship to" section onto a "bill to" section. In HTML forms plus script, the following code would accomplish this:

```
<script type="text/javascript"> <!--
function copyAddresses( ) {
  var frm = document.forms[0];
  frm.shipAddr.value = frm.billAddr.value;
  frm.shipCity.value = frm.billCity.value;
  frm.shipProv.value = frm.billProv.value;
  frm.shipPostCode.value = frm.billPostCode.value;
} --> </script>
```

It would then be activated by a button, with an event-specific attribute, specified like this:

```
<input type="button" id="cp" value="Copy values"
onclick="copyAddresses( )"/>
```

In terms of DOM Level 2 Events, this represents a registration of an observer on the `input` element, watching for the DOM `click` event at the target, and handling the event by calling a short script. As a result, the script in the `onclick` attribute will get called when the user clicks on the button.

Disadvantages of this approach:

- A special **hardwired attribute**, in this case `onclick,` is needed. This is inflexible and clutters the language.
- **Script is difficult to maintain**, especially when bits of script are scattered throughout the document.
- This **won't work in browsers that don't support scripting**.

2.5.2 Declarative Actions

XML Events help specify the same thing as above, declaratively, that is, without the use of scripting. Handlers can come from two sources. XForms defines a number of handlers, called **XForms Actions**, discussed below. Additionally, the host language can define handlers, as is the case with `script`. With XForms Actions, the earlier example can be done without any script at all, like this:

```
<trigger>
  <label>Copy values</label>
  <action ev:event="DOMActivate">
     <setvalue ref="Shipping/Addr"
     value="../Billing/Addr"/>
     <setvalue ref="Shipping/City"
     value="../Billing/City"/>
     <setvalue ref="Shipping/Prov"
     value="../Billing/Prov"/>
     <setvalue ref="Shiping/PostCode"
     value="../Billing/PostCode"/>
  </action>
</trigger>
```

The following section describes all of the XForms Actions defined in XForms. Any of the following can be invoked in such a way that the processing described for the element happens in response to a given event.

- **xforms:delete** Deletes a row of elements from a table. The elements are first deleted from the XForms model, then the table’s repeat deletes the visible items that were linked to those data elements.

- **xforms:insert** Allows you to add a row of elements to a table. This function copies a row of elements in the data model, then inserts the copy in the desired location in the data model. Once the copy is inserted in the data model, the table’s repeat creates corresponding items that are displayed to the user.

- **xforms:message** Sets a message that is displayed to the user in a small dialog box. 3 levels of messages are provided – modal, modeless and ephemeral.
- **xforms:rebuild** Causes the form viewing application to rebuild any internal data structures that are used to track computational dependencies within a particular model.
- **xforms:recalculate** Causes the forms viewing application to recalculate any instance data that is affected by computations and is not up-to-date. This affects all data instances in the designated model.
- **xforms:refresh** Causes the forms viewing application to update all user interface elements linked to a particular model, so that they match the underlying data in the XForms model.
- **xforms:revalidate** Causes the forms viewing application to validate all instance data in a particular model. This ensures that all validation checks have been performed. In general, the XForms processor automatically runs the above 4 actions when required.
- **xforms:reset** Returns a particular XForms model to the state it was in when the form was opened. This allows the user the reset the contents of the form to their ″starting point″, which can increase usability of the form.
- **xforms:send** Triggers an XForms submission. The submission must already be defined in the XForms model.
- **xforms:setfocus** Sets the focus to a particular presentation element in the form.
- **xforms:setindex** Sets the index for the *xforms:repeat* element in a table. This determines which row in the table receives the focus. Rows use one-based indexing. This means that the first row has an index of 1, the second and index of 2, and so on.
- **xforms:setvalue** Sets the value for a specified element in the data model.
- **xforms:toggle** Selects one of the cases in an *xforms:switch* and makes it active. When one case is selected, all other cases in the switch are deselected.

3. XFDL and Other Standards

From 1993 to 1998, UWI.Com developed the Universal Forms Description Language (UFDL). XFDL is the result of developing an XML syntax for the UFDL, thereby permitting the expression of powerful, complex forms in a syntax that promotes application interoperability and adherence to worldwide Internet standards. The current design goals of XFDL are to create a high-level computer language that

1. represents forms as single objects without dependencies on externally defined entities
2. is a human readable plain text
3. is a publicly accessible open standard
4. provides a syntax for inline mathematical and conditional expressions
5. permits the enclosure of an arbitrary size and number of base-64 encoded binary files
6. offers precision layout needed to represent and print dense business/government forms
7. facilitates server-side processing via client-side input validation and formatting
8. permits extensibility including custom items, options and external code functions
9. offers comprehensive digital signature support, including
    a. capture of the whole context of a business transaction
    b. multiple signers
    c. different signers of (possibly overlapping) portions of a form
    d. freezing computations on signed portions of a form

3.1 Language Structure

A digital signature attached to a file accurately identifies the individual who used it, based on the digital certificate provider’s security and the security of the user’s hardware. However, to provide full non-repudiation and auditability, a business transaction not only needs to be signed by someone whose identity is verifiable, it also needs to be representative of the context in which it was signed.

With paper-based forms and documents, this is easily accomplished. Everything that appears on the signed document is considered part of the transaction. **Electronic forms and documents,** however, present a more complex problem in that the exact appearance and functionality of the document must be signed as well as the user's input, or the transaction is meaningless. Legal standards for font size and color must also be observed both when the document is signed and when it is subsequently examined.

Digital signature technology alone can provide the first part of the solution, but not the second. XFDL can be used to create forms that satisfy the second, by presenting a business transaction as a single entity, which is updated as the user fills it in. Item values are stored in XForms instance data, which appears in the same file that contains the user interface and presentation layer markup. When a user digitally signs a form, the XFDL markup for the presentation layer as well as the underlying XForms instance data is signed. Subsequently, when the form is opened in an XFDL viewing or processing application, the current XFDL markup and XForms instance data are compared to those that were used to create the digital signature.

If any discrepancies exist, the signature is flagged as invalid, and the form no longer provides non-repudiation or auditability. Secondary documents can also be placed into an XFDL form as attachments, thus enabling the user to sign both the attachments and the form itself. This method of representing and collecting information in forms and digitally signing and encrypting them ensures that the identity of the signer can be confirmed and that the signer can be proven to have signed the full content and context of the form.

Digital Signatures on text copy of XFDL form = Signature on data, presentation and logic = Transaction Non-Repudiation

XFDL can be said to be divided into the following layers:

1. **Data Layer -** Data layer includes the user data that is stored as XML and the attachments stored in the <data> tag.

2. **Presentation Layer -** This includes the presentation elements.

3. **Presentation Logic** - Presentation elements contain current value and an expression representing logic.

3.2 The Structure of XFDL Forms

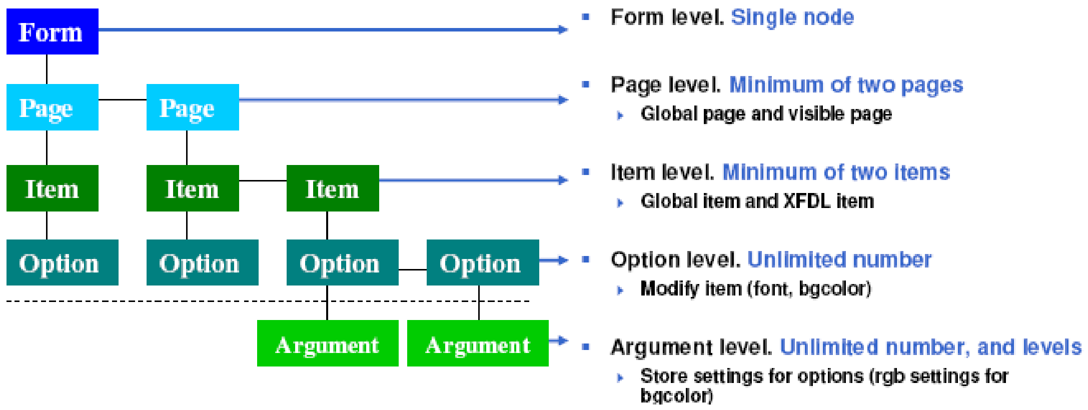

Form
Page
Page
Item
Item
Item
Option
Option
Option
Option
Argument
Argument
Form level. Single node
Page level. Minimum of two pages
Global page and visible page
Item level. Minimum of two items
Global item and XFDL item
Option level. Unlimited number
Modify item (font, bgcolor)
Argument level. Unlimited number, and levels
Store settings for options (rgb settings for bgcolor)

Fig. 3.1 XFDL Document Structure

3.2.1 Top-Level Structure

An XFDL form is an XML 1.0 document whose root element tag is **XFDL**. This element must be in the XFDL namespace,

```
<XFDL xmlns="http://www.ibm.com/xmlns/prod/X 7.1">
...</XFDL>
```

The XFDL element may contain many namespace attributes. By convention, the XFDL namespace is declared to be the default and it is also assigned to the prefix 'xfdl'.The XFDL element must contain a <globalpage> element as the first child element, followed by one or more <page> elements.

```
<!ELEMENT XFDL (globalpage, page+)>
```

The **<globalpage> element** must contain a single <global> element, which can contain zero or more option elements. These are referred to as form global options; they typically contain information applicable to the whole form or default settings for options appearing in the element content of pages. The <globalpage> and <global> elements must contain an attribute called *sid* which must be set to the value global.

```
<!ELEMENT XFDL (globalpage, page+)>
```

```
<!ELEMENT global (%options;*)>
<!ATTLIST globalpage sid CDATA #REQUIRED #FIXED "
global">
```

The Root XFDL Element

```
<?xml version="1.0"?>

<XFDL
xmlns="http://www.PureEdge.com/XFDL/6.0"
>

   <!-- Form Global Options Contain
          1) XML instances in any Schema
          2)  Bindings  to  presentation
layer -->

   <!-- Page(s) of Presentation Layer --
>

</XFDL>
```

A **<page> element** contains a <global> element followed by zero or more 'item' elements. The options in the page's global element typically contain information applicable to the whole page or default settings for options appearing within element content of items. The page global options take precedence over form global options. A page is also required to have a 'sid' attribute, which provides an identifier that is unique among all <page> elements (sid is short for scope identifier).

```
<!ELEMENT page (global, %items;*)>
```

3.2.2 XFDL Items

An item is a single object in a page of a form. Some items represent GUI widgets, such as buttons, check boxes, popup lists, and text fields. Other items are used to carry information such as attached word processing documents or digital signatures.Each item must have a **sid attribute**, which provides a scope identifier that uniquely identifies the item from among all child items of its parent element.

An item can contain zero or more option elements. The options define the characteristics of the item. XForms user interface controls appear as options of XFDL items, and **the XFDL item is said to be the skin of the XForms form control** that it contains. XFDL allows elements in custom namespaces to appear at the item level (as long as they contain an xfdl:sid attribute. Following are the various XFDL Items :

- ***action* -** A non-visible item that can perform similar tasks to a button (print, cancel, submit, and so on) either after a certain period of time or with a regular frequency.
  Skin for: <xforms:submit>, <xforms:trigger>
- ***box* -** An item that provides a graphic effect used to visually group a set of the GUI widgets on the page. A box is drawn under all widgets on a page. This item is useful in some circumstances, but it is usually better to use a pane item (see below) to both visually and logically group related user interface elements.
- ***button* -** Performs one of a variety of tasks when pressed by the user, such as saving, printing, canceling, submitting, digitally signing the form, viewing documents enclosed in the form, and so on. A button can have a text or image face.
  Skin for: <xforms:submit>, <xforms:trigger>, <xforms:upload>
- ***check* -** Defines a single check box.
  Skin for: <xforms:input>
- ***checkgroup* -** Defines a group of checkboxes that operate together to provide a multiselection capability.
  Skin for: <xforms:select>, <xforms:select1>
- ***combobox* -** An edit field combined with a popup list; its value can be either selected or typed.
  Skin for: <xforms:select1> (select or type input), <xforms:input> (date selector)
- ***data* -** Used to carry binary information using base-64 encoding and compression, such as enclosed files or digital images, using base-64 encoding. This item appears when advanced XFDL enclosure mechanisms are used. When a basic <xforms:upload> is used, the data appears in an <xforms:instance> data node.
- ***field* -** Used to capture single- or multiple-line textual input from the user; it includes input validation and formatting features as well as enriched text capabilities.
  Skin for: <xforms:input> (single-line text), <xforms:secret> (single-line, write-only), <xforms:textarea> (for multiline plain text or enriched text)

- ***label* -** Shows either an image or a single or multiple line text value.
  Skin for: <xforms:output>
- ***line* -** A simple graphic effect used as a separator.
- ***list* -** Shows a list box populated with choices from which the user may select one.
  Skin for: <xforms:select>, <xforms:select1>
- ***pane* -** Provides an hierarchic grouping capability for other items that are defined within the content of the pane. Also, may provide the ability to switch between multiple groupings.
  Skin for: <xforms:group>, <xforms:switch>
- ***popup* -** Shows either the text of the currently selected choice or a label if there is no selection; the popup provides a small button that causes the list of selectable choices to appear, from which the user may select one.
  Skin for: <xforms:select1>
- ***radiogroup* -** Defines a group of radio buttons. Initially none may be selected, but a maximum of one radio button can be selected within the group.
- ***signature* -** Receives the signature that ultimately results when a user presses a signature button.
  Skin for: <xforms:select1>
- ***slider* -** Creates a sliding control, similar to a volume control, that lets the user set a value within a specific range.
  Skin for: <xforms:range>
- ***spacer* -** An invisible GUI widget that facilitates spacing in the relational positioning scheme.
- ***table* -** Provides a template of XFDL items that are to be duplicated according to the amount of data available to be displayed. This item provides the ability to dynamically adjust the form rendition based on the amount of data and the amount of changes to that data.
  Skin for: <xforms:repeat>
- ***toolbar* -** Items associated with a toolbar item appear in a separate window pane above the pane for the form page; it is the typical location for page switching and other buttons as its contents are not printed if the form is rendered on paper.

3.2.3 XFDL Options

An option defines a named property of an item, page, or form. Options can appear as form globals, page globals, or as the contents of items.

- ***acclabel* -** Provides a special description of input items that is read by screen reading software.
- ***active* -** Specifies whether an item is active or inactive. In XFDL items containing XForms controls, the default for this option is set by the relevant model item property.
- ***bgcolor, fontcolor, labelbgcolor, and labelfontcolor*** - Specify the colors for an item or its label using either predefined names or RGB triplets in decimal or hexadecimal notation.
- ***border and labelborder* -** Control whether an item or its label has a border.
- ***colorinfo*** - Records the colors used to draw the form when the user signs the form.This is only necessary when the operating system colors are used instead of the colors defined in the form (which is a feature for users with vision impairments).
- ***coordinates*** - Receives the location of a mouse click on an image, if the image is in a button.
- ***cursortype*** - Displays different cursor icons when the user hovers over a button.
- ***data and datagroup*** - Used to create an association between data items and the buttons that provide file enclosure functionality.
- ***delay* -** Used in an action item to specify the timing for the event and whether it should be repeated.
- ***excludedmetadata*** - Used to store special information that is automatically excluded from signatures.
- ***filename and mimetype*** - Give additional information about an enclosed document.
- ***fontinfo and labelfontinfo*** - Defines the typeface, point size, and special effects (bold, italics, and underline) for the font used to display the item's value or label.
- ***format* -** Contains sub-elements that parameterize input validation for the item's value.
- ***formid* -** Defines a unique identifier for the form, such as a serial number.

- ***image* -** Identifies the data item containing the image for the button or label.
- ***imagemode* -** Specifies the display behavior of the image within the data item; the image may be clipped, resized, or scaled to fit the item.
- ***itemlocation, size and thickness* -** Help to define the location and size of the item.
- ***justify* -** Controls whether text in the item should be left, center, or right justified.
- ***label* -** Associates a simple text label with the item; labels can also be created independently with a label item.
- ***linespacing* -** Adjusts the spacing between lines of text in an item.
- ***mimedata* -** Used to store large binary data blocks encoded in bas-64 gzip compressed or base-64 format.
- ***next and previous* -** Link the item into the tab order of the page.
- ***padding* -** Defines how much extra whitespace is put around the pane item.
- ***pageid* -** Defines a unique identifier for a page, such as a serial number.
- ***printbgcolor, printlabelbgcolor, printfontcolor, and printlabelfontcolor* -** Provide the ability to set printing colors for each indicated option different from the display colors on the screen.
- ***printvisible* -** Determines whether an item should be visible when the form is printed. Has no effect on the visibility of the item on the screen.
- ***Printsettings* -** Parameterizes the paper rendition of a form.
- ***readonly* -** Sets the item to be readonly. In XFDL items containing XForms controls, the default for this option is set by the readonly model item property.
- ***rowpadding* -** Defines how much space is applied to the top and bottom of a table row.
- ***rtf* -** Contains the rich text value of rich text fields.
- ***requirements* -** Specifies the requirements for the Web Services to be used by the form.
- ***saveformat and transmitformat* -** Control how the form is written (XFDL, HTML) when it is saved or submitted.
- ***scrollhoriz and scrollvert* -** Control whether a text field item has horizontal and vertical scroll bars or whether it wordwraps, allows vertical sliding, and so on.
- ***texttype* -** Sets whether a field contains plain text or rich text.

- ***transmitdatagoups, transmitformat, transmitgroups, transmititiemrefs, transmititems, transmitnamespaces, transmitoptionrefs, transmitoptions, and transmitpagerefs* -** Work together to allow you to transmit form submissions.
- ***triggeritem* -** Set in the form globals to identify which action, button, or cell activated a form transmission or cancellation.
- ***type* -** Specifies whether the action, button, or cell item will perform a network operation, print, save, digitally sign, and so on.
- ***url* -** Provides the address for a page switch, or for a network link or submission.
- ***value* -** Holds the primary text associated with the item. In XFDL items that contain XForms controls, this option (and all options, such as those that are computed) are treated as transient, which means that any updates to the content are not serialized when the form is written because the updates are reflected in instance data.
- ***visible* -** Determines whether the item should be shown to the user or made invisible.
- ***webservices* -** Defines the nameof the Web Services used by the form.
- ***writeonly* -** Sets the item to be writeonly. This option is only for use with field items that do not skin XForms controls.

3.2.4 Implicit Options

There are some options that are defined within XFDL for the purpose of allowing them to be referenced without being defined by the form author. These options are dynamically added to the document object model (DOM) of the XFDL form while it is being processed, and they are removed when it is serialized. These options tend to be informational in nature or representative of events that can occur while the form is being processed.

- ***activated, focused, and mouseover* -** Indicates whether the form, page or item has been activated or focused or contains the mouse pointer.
- ***dirtyflag* -** In the form global item, this option indicates whether the end-user of the form viewing program has changed the form.
- ***focuseditem* -** At the page global level, records the scope identifier of the item that currently has the focus.

- ***itemprevious, itemnext, itemfirst, itemlast* -** Used to help create a doubly linked list of items in each page. The *itemprevious* and *itemnext* options occur in each item, and itemfirst and itemlast appear at the page global level.
- ***keypress* -** Records a keypress by the user that was not used as input to an XFDL item. The keypress is propagated upwards to the page and form global items.
- ***pageprevious, pagenext, pagefirst, pagelast* -** Used to help create a doubly linked list of pages in the form. The *pageprevious* and *pagenext* options occur in each page, and *pagefirst* and *pagelast* appear at the form global level.
- ***printing* -** In the form global item, this option indicates whether the form is currently printing.
- ***version* -** Appears in the form global item and defines the version of XFDL used to write the form. It is obtained from the XFDL namespace declaration.

3.3 XFDL and XForms

The guiding principle of the integration between XFDL and XForms is that XFDL is quite literally a skin that provides a sophisticated presentation layer for XForms view controls. This means that the user interface controls suggested by XFDL and all of its options and computes are used to style the XForms controls. The XFDL items interact with XForms view controls to consume the instance data and model item properties (MIPs) exposed by the XForms view controls.

3.3.1 Location of XForms Model

The form global option **xformsmodels** is used to contain all **xforms:model** elements in an XFDL document. For example,

```
<XFDL ... >
    <globalpage sid="global">
      <global sid="global">
         <xformsmodels>
              <xforms:model ...>
              <!-- [schema], instances, binds, submissions, actions -->
              </xforms:model>
               ...
         </xformsmodels>
         ...
      </global>
    </globalpage>
<page sid="P1">
      <global sid="global"> ... </global>
       ...
</page>
</XFDL>
```

3.3.2 XFDL Skin Items

1. label for **xforms:output**

The XFDL item most frequently used to display non-editable text is the **label**, so that is the skin for **xforms:output**.

```
<label sid="MonthlyPayment">
   <xforms:output ref="/loan/mpymt"/>
      <format>
         <datatype>currency</datatype>
      </format>
   <xforms:output>
</label>
```

```
<field sid="LoanPrincipal">
   <xforms:input ref="/loan/principal">
         <xforms:label>Loan
         Principal</xforms:label>
   </xforms:input>
</field>
```

2. field for **xforms:input, xforms:secret and xforms:textarea**

The appropriate XFDL item for typed input is the **field**, which therefore can be the skin for single-line (**xforms:input**), single-line write-only (**xforms:secret**) and multiline **xforms:textarea** XForms controls.

3. popup, list, radiogroup and checkgroup Items Skin xforms:select1

Perhaps the most common user interface control for obtaining a user choice is the popup list (a.k.a. the drop down list). This corresponds to the minimal appearance for the **xforms:select1**, which corresponds to the XFDL **popup** item. However, the form author may desire the compact appearance of an XFDL **list** or the full appearance of either a group of radio buttons (**radiogroup**) or a even a single-selection group of check boxes (**checkgroup**) The skin methodology for these items is the same except for the use of the appearance attribute. The minimal appearance is considered to be the default in XFDL, so the form author need not put the attribute when the skin is a **popup**, but the appearance attribute is required for the compact and full settings, and the setting of the appearance attribute must match the XFDL item skinning the **xforms:select1**.

```
<list sid="Currency">
        <xforms:select1 ref="principal/@currency" appearance="compact">
                <xforms:label>Choose currency</xforms:label>
                <xforms:item>
                     <xforms:label>US Dollars</xforms:label>
                     <xforms:value>en_US</xforms:value>
                </xforms:item>
                <xforms:item>
                     <xforms:label>Canadian Dollars</xforms:label>
                     <xforms:value>en_CA</xforms:value>
                </xforms:item>
        </xforms:select1>
        <itemlocation>
        <after>Principal</after>
        <alignt2t>Principal</alignt2t>
        <expandb2b>Principal</expandb2b>
        </itemlocation>
</list>
```

4. The list and checkgroup Items Skin xforms:select

XFDL offers multiselect controls of both compact and full appearance by skinning **xforms:select** with the XFDL **list** and **checkgroup** items, respectively.

5. slider skins xforms:range

Selection of a value within an integer or floating point range can be accomplished with the **slider** item, which skins the **xforms:range**.

```
<slider sid="OverallRating">
   <xforms:range ref="rating" start="1" end="5" step="1">
       <xforms:label>Overall Rating:</xforms:label>
   </xforms:range>
</slider>
```

6. button skins upload

The **xforms:upload** control allows for a single-file attachment. When skinned by an XFDL **button** item, the user can activate a file selection dialog by pressing the button, and the selected file's content will be placed in the referenced instance data node (base 64 encoded).

```
<button sid="GetThePicture">
   <xforms:upload ref="/data" mediatype="image/*">
      <xforms:label>Press me to attach a picture</xforms:label>
   </xforms:upload>
```

7. button and action skin xforms:trigger and xgorms:submit

The **xforms:trigger** is used to activate a sequence of XForms actions, such as inserting or deleting data, changing the input focus or sending a submission. The **xforms:submit** acts as a shorthand for activating an triggering an **xforms:send** action. The XFDL **button** (and the automatic **action**) skins both of these XForms controls.

```
<!-- The hard way -->
<button sid="Submit">
   <xforms:trigger>
      <xforms:label>Submit Loan</xforms:label>
      <xforms:send ev:event="DOMActivate" submission="S"/>
   </xforms:trigger>
   ...
</button>

<!-- The easy way -->
<button sid="Submit2">
   <xforms:submit submission="S">
      <xforms:label>Submit Loan</xforms:label>
   </xforms:submit>
   ...
</button>
```

For other enclosures, refer to section 4.5 (Default Presentation Options).

3.4 XFA

The XML Forms Architecture (XFA) provides a template-based grammar and a set of processing rules that allow businesses to build interactive forms. At its simplest, a template-based grammar defines fields in which a user provides data.

The open nature of XFA provides a common grammar for describing interactive forms. This common grammar provides a common basis for form-related interactions between form processing applications produced by diverse businesses.

Adobe's earlier forms standard was called **Acroforms.** In 2003, Adobe bought JetForms (then called **Accelio**). XFA standard was given by JetForms and therefore, forms developed using this are called Accelio Forms.

3.4.1 Key Features

XFA forms provide a wide range of key features.

- **Workflow :** Data presentation, data capture and data editing, application front-end, printing.
- **Dynamic interactions :** From interactive, human edited forms with dynamic calculations, validations and other events to server-generated machine-filled forms.
- **Dynamic layout :** Forms can automatically rearrange themselves to accommodate the data supplied by a user or by an external data source, such as a database server.
- **Complexity :** Single-page static forms, dynamic document assemblies based on data content, large production runs containing hundreds of thousands of transactions.

XFA is similar to PDF interactive forms introduced in PDF 1.2, which is also known as **AcroForm,** with the following differences:

- XFA can be used in **XML-based workflows**.
- XFA **separates data from the XFA template**, which allows greater flexibility in the structure of the data supported and which allows data to be packaged separately from the form.
- XFA can **specify dynamically-growing forms.**
- XFA can specify **Web interactions**, such as HTTP and Web Services (WSDL). Such interactions can be used to submit data to a server or to request a server perform a calculation and return the result.
- XFA works with other **XML grammars.**

3.4.2 Major Components of an XFA Form

XFA distinguishes between template and data. The **template** defines presentation, calculations and interaction rules. **Content i**s customer's application data. Though they are often packaged together, template and content are separate entities.

1. XFA Template

XFA template is the XFA subelement that describes the appearance and interactive characteristics of an interactive form. It was designed from the ground up to be an XML-based template language.
XFA follows a declarative model in which elements in an XFA template describe the components of the form. That is, XFA template does not include procedures that draw the objects on a form.

Most people are consumers of forms, rather than producers or designers of forms. Yet, in order for a software product to utilize forms, someone first had to expend a degree of thought and work towards the act of creating a form. This specification is focused on the task of form creation, and it is important to distinguish between the “form” that the creator designs, and the “form” that a consumer handles — they both represent the same form, but at two very different stages in the form's life-cycle. XFA clearly distinguishes between the two stages via the following terminology:

- **Form** — what a person filling out a form works with, which is given life by an XFA processing application such as Acrobat.
- **Template** — what the form designer creates, which represents the potential for a form. A template is a collection of related subforms and processing rules.

Templates are containers of content. XFA Template Specification defines **4 types** of Containers.

- **Draw** – Container of static content. Eg- Lines,Rectangles.
- **Field** – Container of dynamic Content.
- **Area** – Static container of other containers.
- **Subform** – Dynamic container of other containers.

2. XFA Data

Typically, XFA variable content is the customer's XML data, matching the customer's schema. Data could also come from a database, an HTTP POST response, a web service interaction, default data supplied by the template or other source. Often, form data elements are plain text, but may also include rich text and graphics.

XFA defines a data value to be **an XFA name/value pair,** where the value is plain or rich text, or a graphic. Data values may contain nested data values. An XFA name is a string suitable for identifying an object. A valid XFA name must be a valid XML name, with the additional restriction that it must not contain a colon (:) character.

XFA also defines a data group: the provider of structure in the data. Data groups may contain data values and other data groups. As stated above, the data is typically structured according to the customer's schema; data values and data groups are represented as abstract structures, inferred from the customer's data. The abstract structure helps the XFA processing application create an XFA form that reflects the structure and content of the data. This process (**called data binding**).

3.5 Comparison of XFA and XFDL

3.5.1 **Major Elements** in XFA and XFDL

Functionality	**XFA**	**XFDL**
Root Element	XFA	XFDL
Workflow	Template	PAGE
Subsectioning in form	Subform, Area	PANE
Placing Duplicate Element of form at oneplace	Prototype Element	XXX

3.5.2 **Formatting Elements** and available options

Purpose	**XFA**	**XFDL**
Determines whether the item should be shown to the user or made invisible.	**Presence** option Set either visible or Hidden	<visible>on/off</visible>
Display specified size of items at a particular location	Options available for item containers are- X= " " Y=" " W=" " H=" " MinW=" " MinH=" " MaxW=" " MaxH=" "	<itemlocation> <x>…</x> <y>…</y> <width>…</width> <height>…</height> </itemlocation>
Determines if an item is read only or read/write	Lock attribute Set it to – 0- item is not locked from changes 1- 1- item is locked from changes	<readonly>on/off</readonly>
Defines the typeface, point size, and bold, italics, or underline for an item or label	`<Font Typeface= "font name" Size= "font size" Weight= "normal/Bold" Posture= "Normal/Italic" Underline="0|1|2" derlinePeriod="All|Word"> <Fill>.font color..` </Fill></Font>	<fontinfo / labelfontinfo> <fontname>…</fontname> <size>…</size> <effect>bold/italic/normal</effect> </ fontinfo / labelfontinfo > <fontcolor>…</fontcolor>
An element that describes the value of a container object.	Value	Value
An Element that describes the alignment of objects	Align	Justify
Element that Describes the appearance of the rectangular surrounding border of an object's enclosing box	Border	border
An element that describes a color	<Color> or <Fill> <Color>…</Color> <solid/></Fill>	Bgcolor, fontcolor, labelbgcolor, labelfontcolor

Element that offsets an object from an enclosing nominal extent	Margin	XXXXX
An Element that Describes a pattern	<Pattern>	XXXXX

3.5.3 Graphical User Interface Elements

Purpose	**XFA**	**XFDL**
Creates a button item and allows a task (similar to an action) to be associated with the button.	Button	Button
Defines a checkbox	CheckButton	check
Defines a combo box	XXXXX	combobox
Creates list box	ChoiceList	list
Defines the position, window, and items for a toolbar	XXXXX	toolbar
Element that permits the entry of date and/or time data	DateTimeEdit	XXXXX
Indicates the application to select a default interface object.	DefaultUI	XXXXX
Element that provides the means by which a form designer can specify a third party UI	ExObject	XXXXX
Element that permits the entry of numeric data	NumericEdit	<format> <datatype>integer/float </datatype> </format>

Specifies a popup menu	XXXXX	Popup
Stores a digital signature	Signature	signature
Element that permits the entry of passwords	PasswordEdit	secret
Defines a radio button	CheckButton	radio
Allows to enter text	TextEdit	textarea
Defines the text for help associated with a given item	Desc	help
Defines an edit control	Field	field
Defines a label to be used with a control	Caption	label
Defines a single entry in a list, popup or combobox.	XXXXX	cell
For positioning other items	XXXXX	Spacer
Element that allows to present text in form	Text	<field><readonly> on</readonly> </field>

3.5.4 Drawing Elements

Purpose	**XFA**	**XFDL**
Mechanism to draw arc, ellipse or circles	<Arc Hand= “ ” Circular= “ ” RadiusX= “ ” RadiusY= “ ” StartAngle= “ ” SweepAngle= “ ”> </Arc>	XXXXX
Draws a Rectangle	Rectangle	XXXXX
Draws a line	Line	line
Draw a table	Table	table
Creates a rectangular area	Box	box

3.5.5 Other useful Elements

Purpose	XFA	XFDL
An Element that Describes an Event Handler	Event	event
An Element that Describes Calculation for a Field	Calculate	<xforms:bind calculate=" " id=" " nodeset=" "></xforms:bind>
An element that Describes a validation for a form object	Validate	<xforms:bind relevant=" " id=" " nodeset=" "></xforms:bind>
An Element that describes a digital signature operation	Signature	Signature
Element specifies a rule for one type of sequencing operation	Traverse	XXXXX
An Element that describes an arbitrary set of application specific properties.	Extras	XXXXX
An Element that Describes a Link	Link	<button> …… <type>link/replace</type> </button>
An Element that describes a link within the current form	Ref	ref
Defines a task, such as print, cancel, submit, and so on	XXXXX	action
To carry binary information using base-64 encoding	XXXXX	data
Specifies the timing for an event in an action item and whether it should be repeated	XXXXX	delay
Defines tab order for items on a page	Traversal	next and previous
Identifies a data item containing an image for a button or label.	Image Allows to specifes both Image Data and Image Link	Image allows to specifies only Image Data
Specifies a popup menu	XXXXX	popup

4. DESIGN

A number of modifications are made in XForms and XFDL. The new language has the following features –

4.1 XML – Based Language

The WWW Forms technology started with HTML. HTML provided for the presentation but there was no common format for the representation of data. Then came XML which provided a common format for representation of data. Today, all web technologies are being developed around XML. A major criteria for acceptance of new languages is whether it would fit into the XML Workflows. XForms and XFDL are XML based. So, we had to ensure that any modifications we make conform with the XML syntax.

4.2 MVC Model

Our language follows a Model-View-Controller programming model as does XForms. But there is nothing in XForms that directly maps to the controller layer. Some consider the control part as the declarative calculations and validations present under the model tag of XForms. Others may consider the dynamic presentation part as the control because that helps us control the presentation. Still others consider XML Events and XFroms Actions as the control. We are in favour of the first concept and have separated the calculations and validations (which we consider the control) from the model part into the control part.

Under the model tag are the schema and instance data for the form. We provide a separate control tag which has **2 child elements - <calc> and <valid>** for calculations and validations respectively.

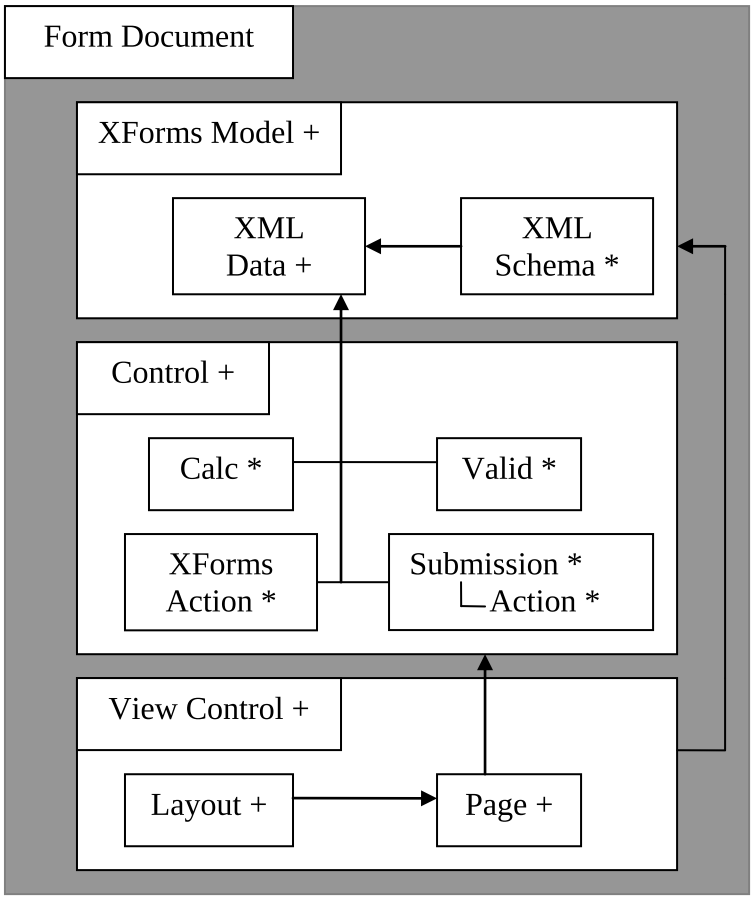


Fig. 4.1 Architecture of the modified Language

4.3 Default Namespaces

The most commonly used namespaces in XForms + XFDL forms need not be specified in our language. As the converter knows which tag belongs to which namespace, it will be assigned the corresponding namespace.

The following namespaces are included into the file by default, others need to be specified. Or if we want to change the prefixes for the existing namespaces, namespace needs to be defined explicitly.

Namespaces

`<XFDL xmlns="http://www.ibm.com/xmlns/prod/XFDL/7.0"`
`xmlns:xforms="http://www.w3.org/2002/xforms"`
`xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom"`
`xmlns:ev="http://www.w3.org/2001/xml-events"`
`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`

4.4 Grouping of similar tags

As seen in the <valid> tag above, variables for which same property needs to be specified are grouped into one <valid>. The same concept in applied throughout the language. This reduces the length of of the code by a huge ratio. The effect can be best seen in real-world forms which have thousands of lines of code. Having a look at such a code reveals there is a large amount of repetition.

To remove this repetitive code, we eg- combine 4 <input> elements into one <input> element and separate the features of each by commas. This leads to a constraint in our language as compared to other XML based languages – the text cannot contain commas. But this problem can be solved by including the comma(,) into the list of prohibited characters for XML and like apostrophe is represented as **&apos:** , comma can be represented as **&com;** (All prohibited characters begin with a ampersand(&) and end with a semicolon(;)).

Example :

In the example below, <xforms:output> is enclosed within <xfdl:label> tag. As specified in the earlier chapters, all XForms tags must be enclosed within XFDL tags for the form to run on a XFDL processor. 4 outputs are specified. In our language, these can be grouped to form one <output> tag.

The resultant reduction in length can be noticed by just having a look at the form. Reducing the size of the code in important for a person who wants to type the form code in a simple textpad or wordpad because tools may not be always available. This will also result in minor reduction in the memory occupied by the form code though this is not a significant factor.

XForms Code

```
<label sid="cnt">
      <xforms:output ref="/purchaseOrder/totals/rowcount">
            <xforms:label>No. of items = </xforms:label>
      </xforms:output>
</label>

<label sid="sub">
     <xforms:output ref="/purchaseOrder/totals/subtotal">
            <xforms:label>Subtotal = </xforms:label>
      </xforms:output>
</label>

<label sid="tax">
      <xforms:output ref="/purchaseOrder/totals/tax">
            <xforms:label>Tax = </xforms:label>
      </xforms:output>
</label>

<label sid="total1">
      <xforms:output ref="/purchaseOrder/totals/total">
            <xforms:label>Total = </xforms:label>
      </xforms:output>
</label>
```

New Code

```
<output path="purchaseOrder/totals">
      <id> cnt,sub,tax,total1 </id>
      <what> /rowcount,/subtotal,tax,total </what>
      <label>No. of items = ,Subtotal = ,
            Tax = ,Total = </label>
</output>
```

4.5 Default Presentation Options

It is tedious to write code in XFDL using XForms because for the form to run on a XFDL processor, each XForms tag must be enclosed within a XFDL tag eg- <xforms:input> may be enclosed within <field>,<check> or <combobox>. But in most of the forms you see, you will find <xforms:input> enclosed within <field>. So, we have taken <field> as the default enclosure for <xforms:input>. If <xforms:input> needs to be enclosed within <combobox>, our language provides a **displayType attribute** on each XForms item. If displayType attribute is not provided, default enclosure is used.

This has been done for all the XForms form controls. The following table indicates the default enclosures for each.

Form Control	**Possible Skins**	**Default Skin**
output	label	label
input	field, check, combobox	field
secret, textarea	field	field
select	list, checkgroup	list
select1	list, popup, combobox, radiogroup, checkgroup	popup
range	slider	slider
upload	button	button
Submit, trigger	button, action	button
group, switch	pane	pane
repeat	table	table

Table 4.1 List of Default Skins

XForms Code	**New Code**
`<check sid="healthPlan_check">` `<xforms:input ref="healthinfo/healthPlan">` `</xforms:input>` `<label>Active Health Plan</label>` `</check>`	`<input displayType="check" id="healthPlan_check" what="healthinfo/healthPlan" label="Active Health Plan>` `</input>`

4.6 Seperation of Layout Information

Layout means the size and location of various items on the page being displayed. As layout information deals with the presentation, it is specified in XFDL and XForms is not concerned with it. In any real-world form coded using XFDL, you will find the **<itemlocation>** tag placed within each and every item of the form.This tag is used to specify the x and y coordinates at which the item must be placed on the form. It may also specify the width and height of the item.

In the language that we propose, we separate this layout information from the actual item specification. That is ,the properties of the items is specified separately and layout separately.The itemlocation is also a property of the items, but as it is common to each item, it can be specified at a separate place.
A XFDL form has a number of pages and therefore, each page will have seperate layout. The information must be specified in the new **<layout>** tag which is the subelement of the <page> element. The example below shows how this can be done. Notice the reduction in size of the code. This is even more for larger forms.

XForms Code

```
<page sid="page1">
      <global sid="global"/>

      <label sid="persInfo_label">
            <value>Personal Information</value>
            <itemlocation>
                  <x>20</x>
                  <y>30</y>
            </itemlocation>
      </label>

      <label sid="name">
            <value>Name : </value>
            <itemlocation>
                  <x>20</x>
                  <y>60</y>
            </itemlocation>
      </label>
      <label sid="name_out">
            <xforms:output ref="info/name">
            </xforms:output>
            <itemlocation>
                  <x>60</x>
                  <y>60</y>
            </itemlocation>
      </label>
</page>
```

New Code

```
<page id="page1">
      <label>
            <id> persInfo_label, name </id>
            <value>Personal Information, Name :</value>
      </label>

      <output id="name_out" what="info/name"/>

      <layout>
            <items>
                  <id> persInfo_label, name, name_out
                  </id>
                  <coord> (20,30), (20,60), (60,60)
                  </coord>
            </items>
      </layout>

</page>
```

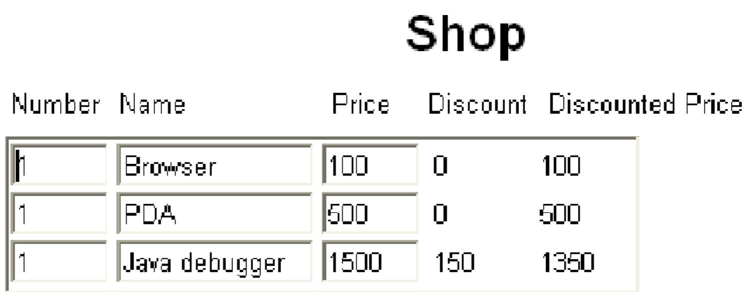
4.7 Modifications in the <table>

Most forms that display or input similar information for more than entity, use tables. Eg- A shop has several items, each having a name, price and may have some discount on it. Suppose, we want to represent this information for 10 items, then it is best done using tables.

XForms supports dynamic presentation through the **repeat** tag, which helps in creating a table whose rows can be dynamically increased or decreased. So, most of the tables seen in real-world forms have 2 buttons at the bottom, one for inserting rows into the table and other for deleting them.

But, XFDL just provides the table tag, and no explicit support for the header row of the table and buttons at the bottom. Even HTML tables have a header row but in XFDL, we have to create the header row using separate label tags as can be seen in the example below.

Example :
For creating the following table, the original code and the new code are shown –



Number	Name	Price	Discount	Discounted Price
1	Browser	100	0	100
1	PDA	500	0	500
1	Java debugger	1500	150	1350

There are **2 possible syntaxes** that we propose for <table> :

- **First code** below has labels for each input and output within the table. These **labels** can be used to provide the header row for the table but remember, these labels appear only once in the table and not as many times as the input or output appears.
- The **second code** shows an explicit **heading tag.**

XForms Code

```
<label sid="label1">
<value>Number</value>
</label>
<label sid="label2">
<value>Name</value>
</label>
<label sid="label3">
<value>Price</value>
</label>
<label sid="label4">
<value>Discount</value>
</label>
<label sid="label5">
<value>Discounted Price</value>
</label>
<table sid="itemtable">
      <xforms:repeat id="repeat1"
      nodeset="/purchaseOrder/items/item">

            <field sid="field1">
                  <xforms:input ref="units">
                        <xforms:hint> The units of this item
                        </xforms:hint>
                  </xforms:input>
            </field>
            <field sid="field3">
                  <xforms:input ref="price">
                      <xforms:hint>The price of this item
                      </xforms:hint>
                  </xforms:input>
            </field>
            <label sid="output1">
                  <xforms:output ref="discount">
                  </xforms:output>
            </label>
            <label sid="output2">
                  <xforms:output ref="total">
                  </xforms:output>
            </label>
</xforms:repeat>
</table>
<button sid="insert">
      <xforms:trigger id="insert1">
            <xforms:label>Add item</xforms:label>
            <xforms:action ev:event="DOMActivate">
                  <xforms:insert at="index('repeat1')"
                  nodeset="/purchaseOrder/items/item"
                  position="after">
                  </xforms:insert>
            </xforms:action>
      </xforms:trigger>
</button>
<button sid="delete">
      <xforms:trigger id="delete1">
            <xforms:label>Delete item</xforms:label>
            <xforms:action ev:event="DOMActivate">
                        <xforms:delete at="index('repeat1')"
                        nodeset="/purchaseOrder/items/item">
                        </xforms:delete>
            </xforms:action>
      </xforms:trigger>
</button>
```

First Code

```
<table>
      <repeat what="purchaseOrder/items/item" id="repeat1">
            <input>
                  <id>unts,nm,pr </id>
                  <label>Number, Name, Price </label>
                  <what>/uints,/name,/price</what>
                  <hint>The units of this item,
                        The name of this item,
                        The price of this item </hint>
      </input>
            <output>
                  <id>discnt,tot</id>
                  <label>Discount, Discounted Price </label>
                  <what>/discount,/total </what>
            </output>
      </repeat>
      <trigger>
            <id> insert1, delete1 </id>
            <label> Add item, Delete item </label>
            <type> insert, delete </type>
      </trigger>
</table>
```

Second Code

```
<table>
      <heading>
            <id>no1,name1,price1,dis1,disprice1</id>
            <value>Number,Name,Price,Discount,Discounted
            Price</value>
      </heading>
      <repeat what="purchaseOrder/items/item" id="repeat1">
            <input>
                  <id>unts,nm,pr</id>
                  <what>/uints,/name,/price</what>
                  <hint>The units of this item,
                        The name of this item,
                        The price of this item
                  </hint>
            </input>
            <output>
                  <id>discnt,tot</id>
                  <what>/discount,/total</what>
            </output>
      </repeat>
      <trigger>
            <id> insert1, delete1 </id>
            <label> Add item, Delete item </label>
            <type> insert, delete </type>
      </trigger>
</table>
```

Both the tables contain a trigger tag for the insert and delete buttons at the end of the table. Instead of creating separate buttons, we have included this within the table. The **<type> element** specifies the type of the button – whether it is for insertion or deletion. These types are introduced by us and are not build into XFDL.

4.8 The <select> clause

There are **2 clauses in XForms – select and select1**, to specify selection of one or more and one item respectively. We have combined these into one clause that is **<select> with a type attribute** whose possible values are **one or many**. The default value for the type attribute is many that means, the <select> without the type attribute is equivalent to XForms select.

Secondly,XFroms provides **2 versions of select** – one contains **the itemset** element that refers to the instance data and the in the other, **items** are explicitly specified. Following examples show the new code for each of these.

Example (1) :

XForms Code

```
<checkgroup sid="currency">
      <xforms:select ref="currency" appearance="full">

      <xforms:label >Select the currencies you accept:
      </xforms:label>

      <xforms:item>
            <xforms:label>US Dollars </xforms:label>
            <xforms:value> USD </xforms:value>
      </xforms:item>

      <xforms:item>
            <xforms:label>CDN Dollars </xforms:label>
            <xforms:value> CDN </xforms:value>
      </xforms:item>

      <xforms:item>
            <xforms:label>Euro</xforms:label>
            <xforms:value>Euro</xforms:value>
      </xforms:item>

      </xforms:select>
</checkgroup>
```

New Code

```
<select id=”currency” displayType =”checkgroup”
what=‟currency”>

      <label> Select the currencies you accept:</label>
      <items>
            <label>US Dollars, CDN Dollars, Euro </label>
            <value> USD, CDN, Euro </value>
      </items>

</select>
```

Example (2) :

XForms Code
```
<checkgroup sid="currency">
      <xforms:select ref="currency" appearance="full">
            <xforms:label>Select the currencies you accept:
            </xforms:label>

            <xforms:itemset
            nodeset="instance('currency')/choice">
                  <xforms:label ref="@show"> </xforms:label>
                  <xforms:value ref="."> </xforms:value>
            </xforms:itemset>
      </xforms:select>
</checkgroup>
```

New Code
```
<select id="currency" what="currency">
      <label>Select the currencies you accept: </label>

      <items what="instance('currency')/choice">
                  <label what="@show"> </label>
                  <value what="instance('currency')/choice">
                  </value>
            </items>
      </select>
```

4.9 The use attribute

A reduction in size of the form can be provided by defining something common, which can be reused again and again. This helps the developer as he need not repeat the same things again and again. The use attribute suffixes to do this.

The new attribute called **use** helps us to use a preexisting feature. We can specify some common features of the form in the globals section of the form or the globals section of the page, which can be inherited using the use attribute. Eg – A action may have to be repeated again and again, so we define it in the globals section and whenever it is required, we can refer to it through the use attribute.

The globals and pageglobals are the only 2 places where we can define the common features, not anywhere else in the form. Also,this can be done only for inbuilt XFDL tags and options. That is, we cannot define a new tag. New tag concept is already provided in XFDL through custom namespaces.

The common feature is given an id which is then refered to by the use attribute. The new feature forms a kind of prototype. The options in this prototype can be overridden by specifying new options at the place of reference.Eg- if the labels have the same fontname and size, we can define the fontinfo in the globals section. But different labels may have different effects (some may be bold and others italic), so this feature can be overridden.

Example :

New Code

```
<global>
        <fontinfo id="fnt">
                <fontname>Arial</fontname>
                <size>8<size>
                <effect>italic</effect>
        </fontinfo>

        <fontinfo id="fnt1" use=”fnt”>
                <effect>bold</effect>
                </fontinfo>
        </fontinfo>

</global>


<output id= "LABEL1">
        <label>Client Information</label>
         <fontinfo use="fnt1">
              <size>10</size>
         </fontinfo>
</output>
```

In the example above, we define a prototype font with the id of **“fnt**”. Once defined in the globals section, these font characterstics can be used anywhere in the form by just referring to the id fnt. We can define a new font prototype in terms of the first, as shown above.

As mentioned earlier, some properties can be overridden by specifying a new property at the place of use. In the example above, the fnt specifies a font size of 8 which is inherited by fnt1. But output field specifies font size of 8 which will override the font size mentioned in the prototype.

4.10 Additional Functions

XFDL provides a number of function. We added some new ones :

(I) String Functions

concat(str1, str2) – To concatenate 2 strings.

str(number) – converts a number to a string.

(II) Math Functions

max(number1, number2, ….) – Maximum of a set of numbers.

min(number1, number2, ….) – Minimum of a set of numbers.

avg(number1, number2, …) – Average of a set of numbers.

count(number1, number2, …) – Number of elements in a set.

atan2(numerator,denominator) – the arc tangent of quotient of 2 given numbers.

(III) Financial Functions – XFDL being a language for E-commerce forms must have some inbuilt financial functions.

Functions for simple interest

interest(principle, rate, period)

principle(interest,rate,period)

rate(interest,principle,period)

period(interest,principle,rate)

Functions for compound interest

cinterest(principle, rate, period)

cprinciple(interest,rate,period)

crate(interest,principle,period)

cperiod(interest,principle,rate)

5. IMPLEMENTATION

5.1 XML Parsers

A *parser* is a piece of program that takes a physical representation of some data and converts it into an in-memory form for the program as a whole to use. An *XML Parser* is a parser that is designed to read XML and create a way for programs to use XML. The main types of parsers are : **SAX, DOM and pull.**

1. SAX

SAX stands for ***Simple API for XML***. Its main characteristic is that as it reads each unit of XML, it **creates an event that the calling program can use**. This allows the calling program to ignore the bits it doesn't care about, and just keep or use what it likes. The disadvantage is that the calling program must keep track of everything it might ever need.

2. DOM

DOM (***Document Object Model***) is an official recommendation of the W3C. It differs from SAX in that it builds the ***entire*** **XML document representation in memory** and then hands the calling program the whole chunk of memory. DOM can be very memory intensive as the entire tree has to be stored in the memory.
line with the philosophy of the local language. Examples in Java include TinyTree (used only in Saxon), JDOM, DOM4J and XOM.

3. Pull Parser

SAX is a *push* parser, since it pushes events out to the calling application. *Pull* parsers, on the other hand, sit and wait for the application to come calling. They ask for the next available event, and the application basically loops until it runs out of XML.

It is designed to be used with large data sources, and unlike SAX which returns *every* event, **the pull parser can choose to skip events** (or in some implementations, whole sections of the document) that it is not interested in. The adapters are designed to work with both the SAX and the pull parser interfaces.

	SAX	**DOM**
Origin	Previously result of xml-dev community, started as Java only interface. Now maintained by the SourceForge organization.	W3C Organization recommendation. DOM is not an API.
Interface type	Primarily a java interface only. Now interfaces available on most programming languages	Language and platform – neutral recommendation.
Resource consumption	Limited impact	High impact on memory and processing resource as the DOM create in-memory representation of the XML document
How it operates	Event based interface. Events are triggered where the SAX parser encounters an XML tag.	Parses the XML document first, and then creates an in-memory representation of XML file as a nodes tree.
Document handling	Read-only parser	Can manipulate nodes and add , delete nodes.
Examples of Ideal situations for use	When memory/processing power is restricted. When XML documents size is very large and no random access is required.	When Random access of XML documents is required. For manipulation of in-memory structure of an XML document. When support for Namespaces is desirable.

Table 5.1 A Comparison of the SAX and DOM Parsers

Another way that parsers are classified is - Validating versus non-validating parsers.**Validating parsers** validate XML documents as they parse them, while **non-validating parsers** don't. In other words, if an XML document is well-formed, a non-validating parser doesn't care if that document follows the rules defined in a DTD or schema, or even if there are any rules for that document at all.

There are 2 reasons that we use a non-validating parser :

- **Speed and efficiency**. It takes a significant amount of effort for an XML parser to read a DTD or schema, then set up a rules engine that makes sure every element and attribute in an XML document follows the rules.
- **If you're sure that an XML document is valid** (maybe it's generated from a database query, for example), you may be able to get away with skipping validation. Depending on how complicated the document rules are, this can save a significant amount of time and memory.

How to use a parser

Generally, the following 3 steps are involved in programs using parsers :

1. Create a parser object
2. Point the parser object at your XML document
3. Process the results

5.2 The DOM Parser

When you parse an XML document with a DOM parser, you get a hierarchical data structure (a DOM tree) that represents everything the parser found in the XML document. You can then use functions of the DOM to manipulate the tree. You can search for things in the tree, move branches around, add new branches, or delete parts of the tree.

From a Java-language perspective, **a Node is an interface**. The Node is the base datatype of the DOM; everything in a DOM tree is a Node of one type or another. DOM also defines a number of subinterfaces to the Node interface:

- **Element :** Represents an XML element in the source document.
- **Attr :** Represents an attribute of an XML element.
- **Text :** The content of an element. This means that an element with text contains text node children; the text of the element is *not* a property of the element itself.

- **Document :** Represents the entire XML document. Exactly one Document object exists for each XML document you parse.

Additional node types are: Comment, ProcessingInstruction, and CDATASection, which represents a CDATA section.

5.2.1 DOM APIs

JAXP, the Java API for XML Parsing specifies certain common tasks that the DOM and SAX standards leave out. Specifically, creating parser objects is not defined by the DOM or SAX standards.

The Document Object Model implementation is defined in the following packages:

- **org.w3c.dom -** Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.

- **javax.xml.parsers -** Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. This package also defines the ParserConfigurationException class for reporting errors.

We can use the DocumentBuilder newDocument() method to **create an empty Document** that implements the org.w3c.dom.Document interface. Alternatively, we can use one of the builder's parse methods to create a Document from existing XML data.

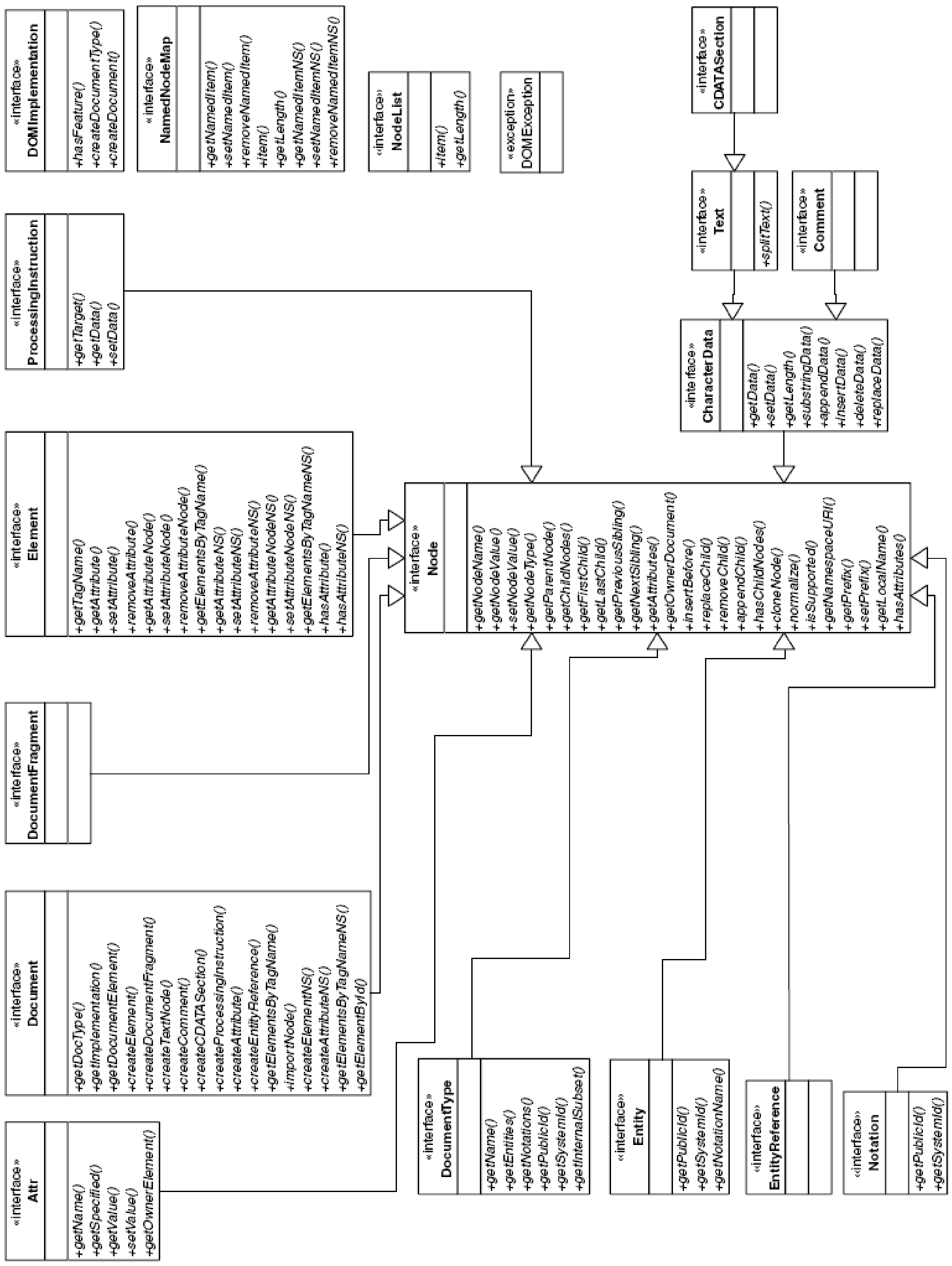


Fig. 5.1 The DOM Interface Hierarchy

5.2.2 Reading XML Data into a DOM

The following steps are involved in creating a DOM from an existing XML file :

1. Import the Required Classes

These lines import the JAXP APIs that we will be using :

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

These lines import the exception details for exceptions that can be thrown when the XML document is parsed. DOMExceptions are only thrown when traversing or manipulating a DOM. Errors that occur during parsing are reported using a same mechanism as SAX :

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

Finally, import the W3C definition for a DOM and DOM exceptions:

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

2. Declare the DOM

The **org.w3c.dom.Document** class is the W3C name for a Document Object Model (DOM). Whether we parse an XML document or create one, a Document instance will result.

static Document document;

3. Handle Errors

Next, we put in the error handling logic. The error-handling code for DOM and SAX applications are very similar:

try {
} **catch (SAXException sxe)** {
// Error generated during parsing
Exception x = sxe;
if (sxe.getException() != null)

```
x = sxe.getException();
x.printStackTrace();
} catch (ParserConfigurationException pce) {
// Parser with specified options can't be built
pce.printStackTrace();
} catch (IOException ioe) {
// I/O error
ioe.printStackTrace();
}
```

4. Instantiate the Factory

Next, we add the code highlighted below to obtain an instance of a factory that can give us a document builder:

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

5. Get a Parser and Parse the File

DocumentBuilder builder = factory.newDocumentBuilder();
document = builder.parse(new File(argv[0]));

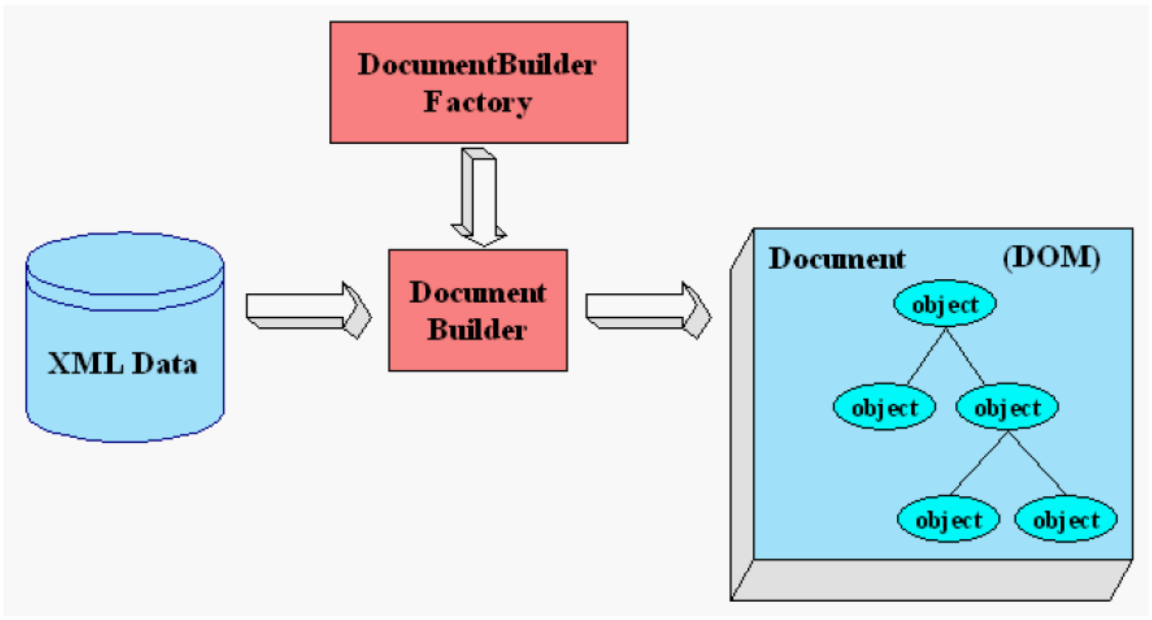


Fig. 5.2 Steps in creating a DOM Tree

6. Setting DOM parser features

After getting the document object, we may want to configure the factory. The most important methods are:

setValidating(boolean) - Sets the factory's validation property.
isValidating() - Returns true if the factory creates validating parsers, false otherwise.
setNamespaceAware(boolean) - Sets the factory's namespace-aware property.
isNamespaceAware() - Returns true if the factory creates namespace-aware parsers, false otherwise.
setIgnoringElementContentWhitespace(boolean) - Sets the factory's whitespace property. If this is true, the parsers created by the factory won't create nodes for the ignorable whitespace in the document.
isIgnoringElementContentWhitespace() - Returns true if the factory creates parsers that ignore whitespace, false otherwise.

5.2.3 Creating a new DOM

We are still going to create a document builder factory, but this time you're going to tell it create a new DOM instead of parsing an existing XML document.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
DocumentBuilder builder = factory.newDocumentBuilder();
document = builder.newDocument();
```

In this code, you replaced the line that does the parsing with one that creates a DOM.

And since we are going to be working with **Element objects**, we add the statement to import that class at the top of the program:

```
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import org.w3c.dom.Element;
//create an element XFDL and its children
Element root = (Element) document.createElement(“XFDL");
document.appendChild(root);
root.appendChild( document.createTextNode("some") );
root.appendChild( document.createTextNode(" ") );
root.appendChild( document.createTextNode("text") );
```

For more details on DOM methods, refer [20] .

5.3 The Conversion

The design chapter describes the modifications made to the XForms and XFDL. Our implementation part includes the conversion of this code into the XFDL code. The final form can be displayed in the IBM Workplace Forms Viewer which supports XForms + XFDL.

For the conversion, we have used the **Java Apache Xerces Parser** which comes with Java v1.4 and above. For lower versions of java, it can be integrated by downloading it from http://xerces.apache.org.

It is a fully conforming XML Schema processor.

- ✓ Supports Simple API for XML (SAX) 2.0.2
- ✓ Supports Document Object Model (DOM) Level 3
- ✓ Support XML 1.0 and Namespaces in XML 1.1 Recommendation

The conversion involves the following **steps :**

1. Read in the modified code and create a DOM Tree from it, by using a DOM Parser as decribed above.

2. This DOM Tree is subjected to the conversion code, which performs the appropriate conversion. The result is a new DOM tree.

3. The final DOM tree is then serialized to get an XML document (actually a XForms + XFDL document).

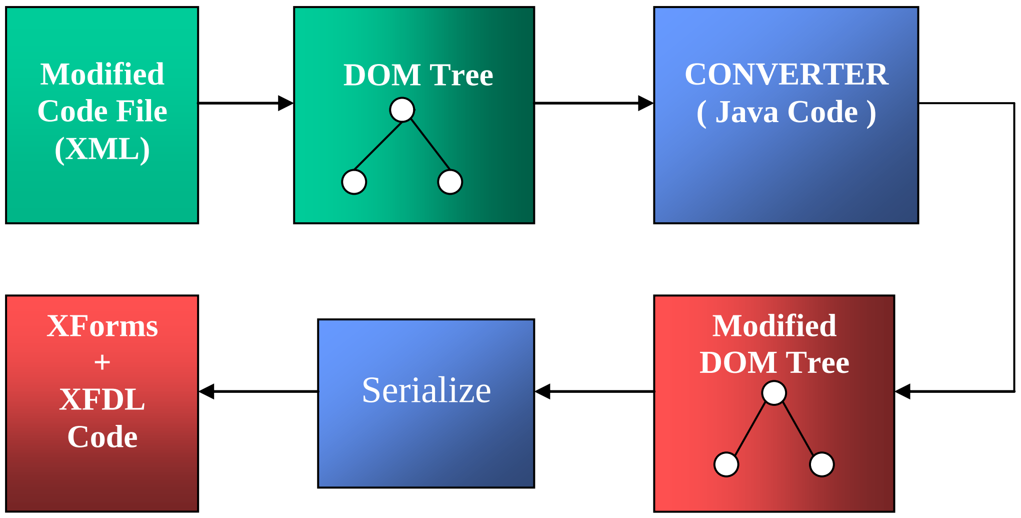


Fig. 5.3 Conversion Steps

5.4 Architecture of the converter

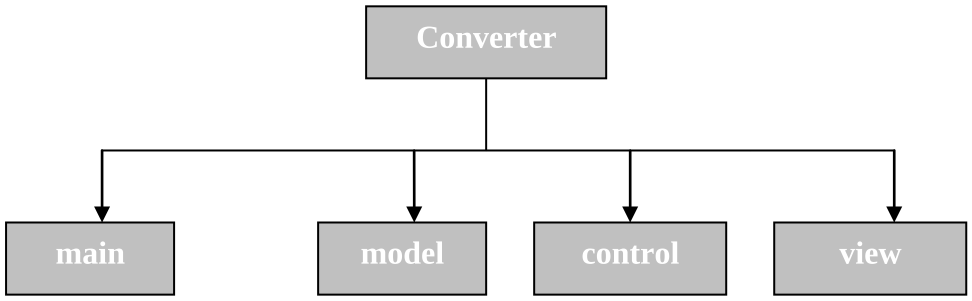


Fig. 5.4 Parts of the converter

As can be seen from the figure above, the converter consists of 4 major parts (coded in java as 4 separate packages).

5.4.1 The main package

This package contains the beginning code for the converter that is, code for parsing the XML file and create DOM tree. Also, it contains code to create the final XFDL document. It contains the following classes.

(i) convert – parses our modified code file and creates the DOM tree. It instantiates the createXFDL, createModelObject and createViewObject classes.

(ii) createXFDL – This class creates a new Document object, adds root element to it, and prints this to a XFDL file. It also sets the namespace attributes on the root element which specifies all the namespaces that can be used within the document.

(iii) separate - This class contains the code for converting the comma-seperated list of parameters into individual elements and appends the value of path to relative parameters. It is placed in the main class as it is a common class instantiated by various other classes. The code present here helps us to separate the similar tags that we did combine in our language

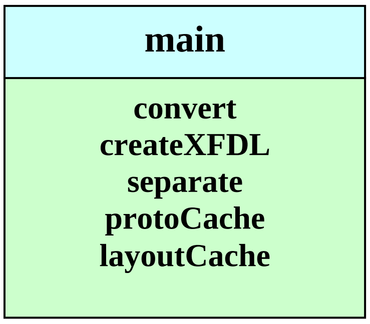


Fig. .5.5 The main package

(iv) prototype Cache – protoCache is used to store the prototype information. Whenever a new prototype is encountered in the globals section of the form, an instance of this class is created and it stores the prototype for later reference by any of the form controls.

(v) layout Cache – layoutCache reads the layout information within the <layout> tag and creates a cache that stores the ids and the corresponding location and size in formation. Each form control can call this class to obtain its location information.

5.4.2 The model package

This package contains the model part of the MVC paradigm. It contains the classes shown in the figure below :

(i) createModelObject – This class reads in the model elements and their properties from the modified code file, and creates the model objects. It also instantiates the createControlObject and createModelElement classes.

(ii) model – This class stores the model item properties namely model id, data file name or the actual data and schema.

(iii) createModelElement – This class takes the model object as input, and creates a model Element from it for storage in the new file. It instantiates the createBindSubmissionElement class as in the final XFDL file, bind and submission elements are part of the model itself.

(iv) createInstanceObject – A model can have various instances that represent the data instance. Each instance may be identified by an id. In our language, within the instance tag, we may give the file name or the actual data. This class reads the data within the <data> tag and stores it within the instance object.

(v) instance – This class stores the instance properties like nodelist of the children and the id.

(vi) createInstanceElement – This class creates the appropriate instance tag from the instance object. It checks if the child node of instance element is a text node. If it is, that means the user has given a file name which contains the instance, so it copies the file name into the src attribute. If the child is an element node, it imports all the children as such.

In the same way, there are 3 classes for handling the schema namely, createSchemaObject, schema, createSchemaElement.

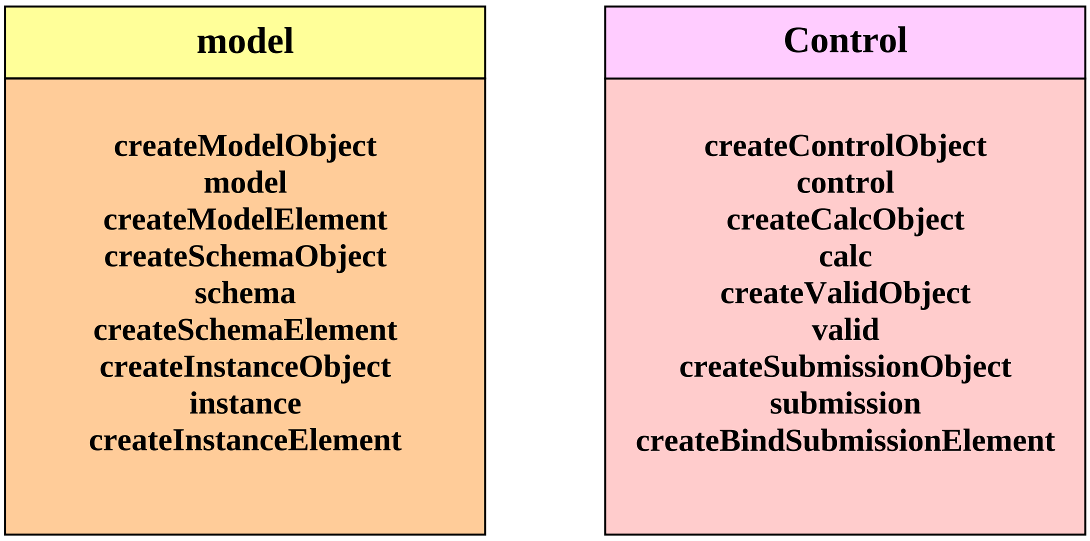


Fig. 5.6 model and control packages

5.4.3 The control package

It contains the code for **calculations, validations and submissions**. All these elements are converted to binds. It contains the following classes. The classes provide the similar functionalities as the corresponding classes in the model package. That is, createControlObject creates the control object that stores the control properties. As the name signifies, createBindSubmissionElement creates the bind elements from calc and valid elements and submission elements from the corresponding submission elements.

The Encryption handler

One modification made is the option for submission of encrypted data. In the encryption element, the user can mention encryption method to use. We use java inbuilt functions for the encryption. Whenever the submit button is clicked and encryption is set, the handler corresponding to the encryption is activated and data is submitted in the encrypted form. The handler is of course provided on the submit event.

5.4.4 The view package

This package contains a number of sub-packages for each specific presentation element. Following figure shows the view package, its classes and its sub-packages.

View contains the **following classes –**

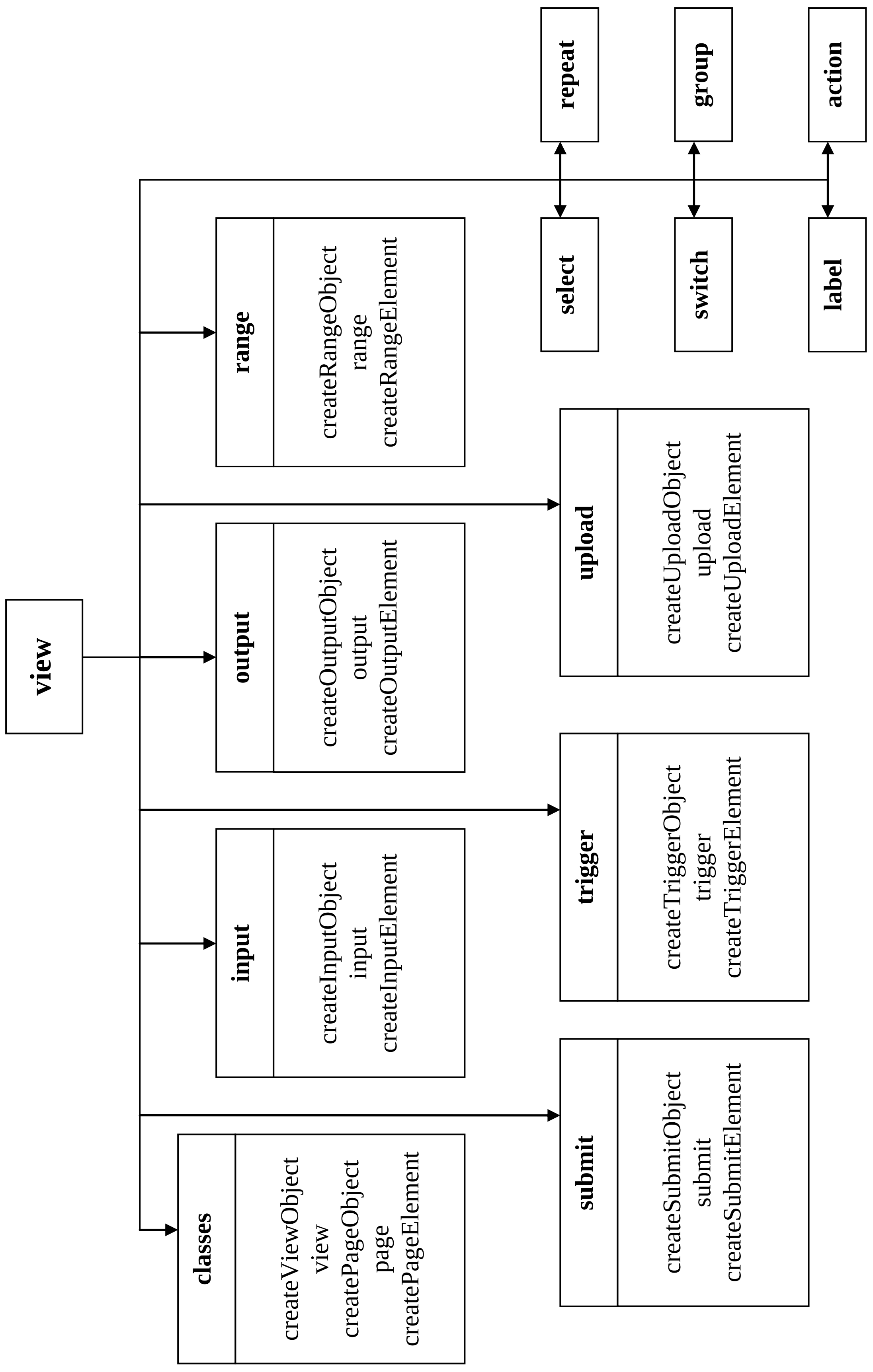


Fig. 5.7 The view Package

- createViewObject
- view
- createPageObject
- page
- createPageElement

There is no view element present in the XFDL file. The packages **input, output, range, submit, trigger and upload** have been specified in detail. The other packages namely **action, select, switch, repeat, group** further contain sub-packages which are depicted in separate package diagrams.

1. **The select package –** It contains the 3 files for creating the select element itself and a **package items** for creating the item or itemset element from the items element. Items further calls the label,value and copy classes.

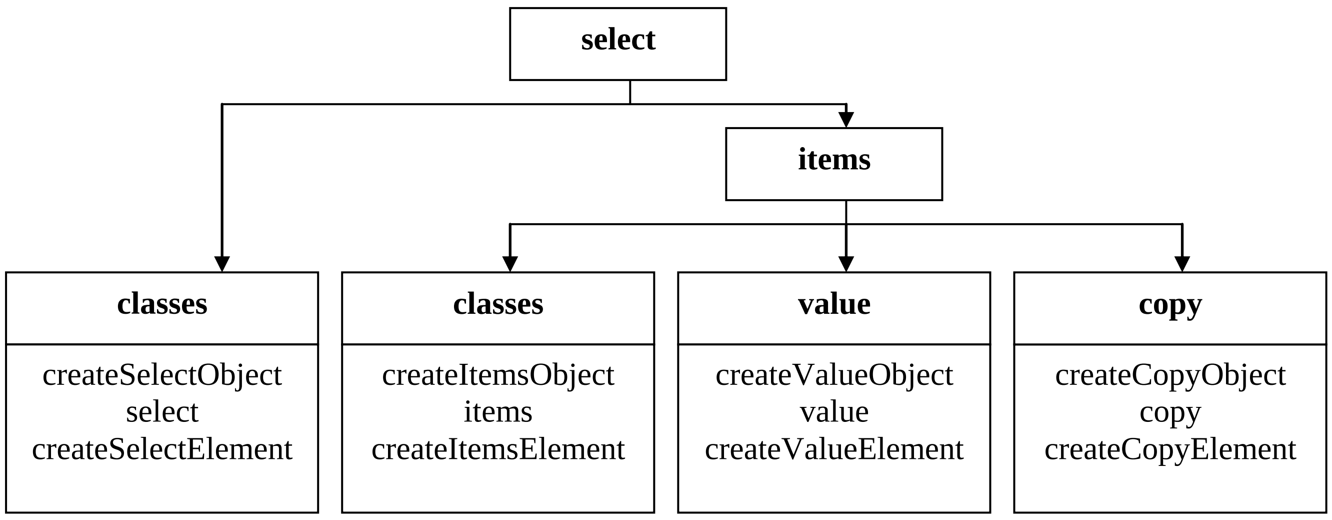


Fig. 5.8 The select package

2. **The pane package** – An XFDL pane may contain xforms:switch and xforms:group. The switch inturn contains case and its classes. Group module may call any of the form controls.

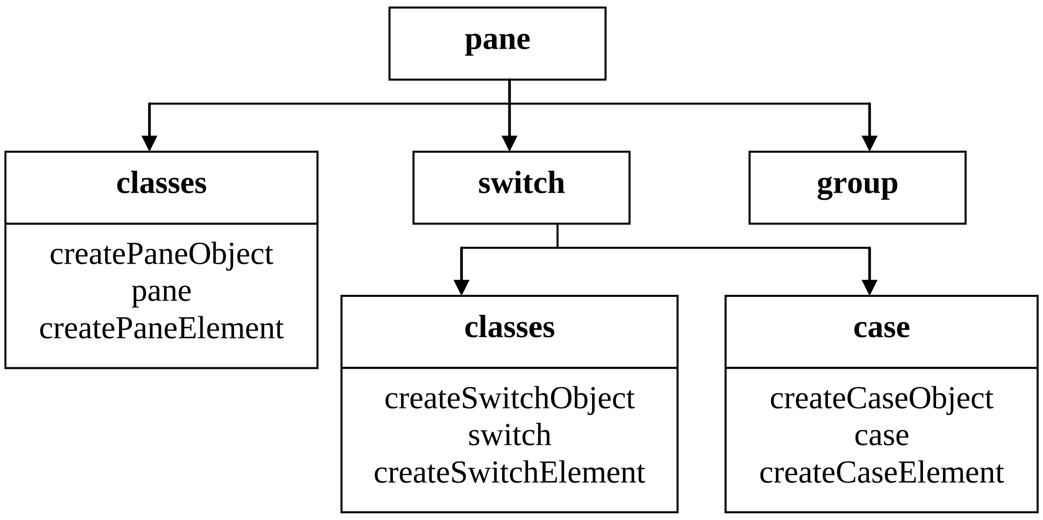


Fig. 5.9 The pane package

3. **The table package –** The table package contains classes for drawing the table, extracting the heading form the labels, creating the repeat construct and the default buttons – add and delete.

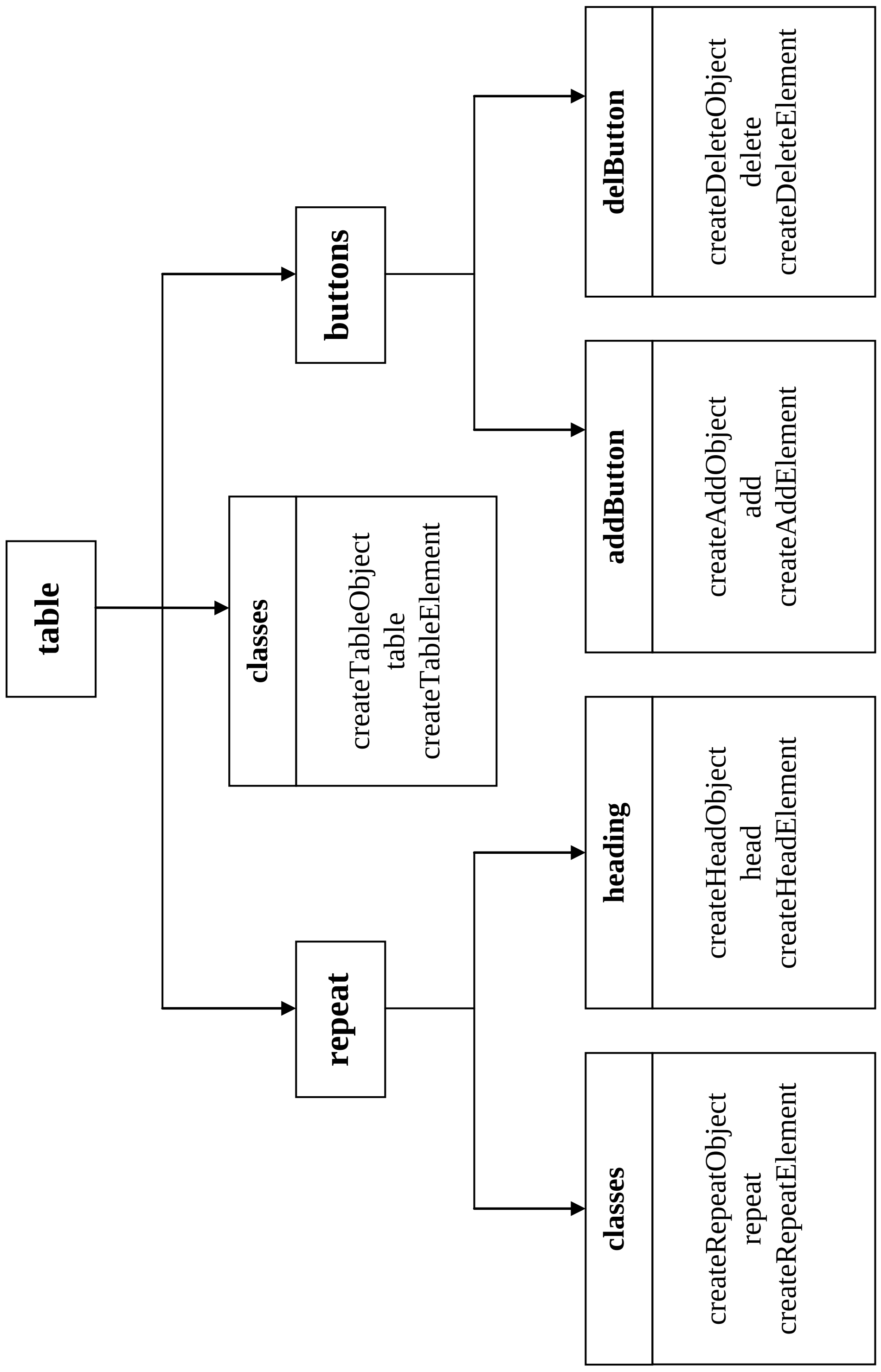


Fig. 5.10 The table Package

3. **The action package –** The action package contains the following classes – createActionObject, action, createActionElement and the subpackages as mentioned in the figure.

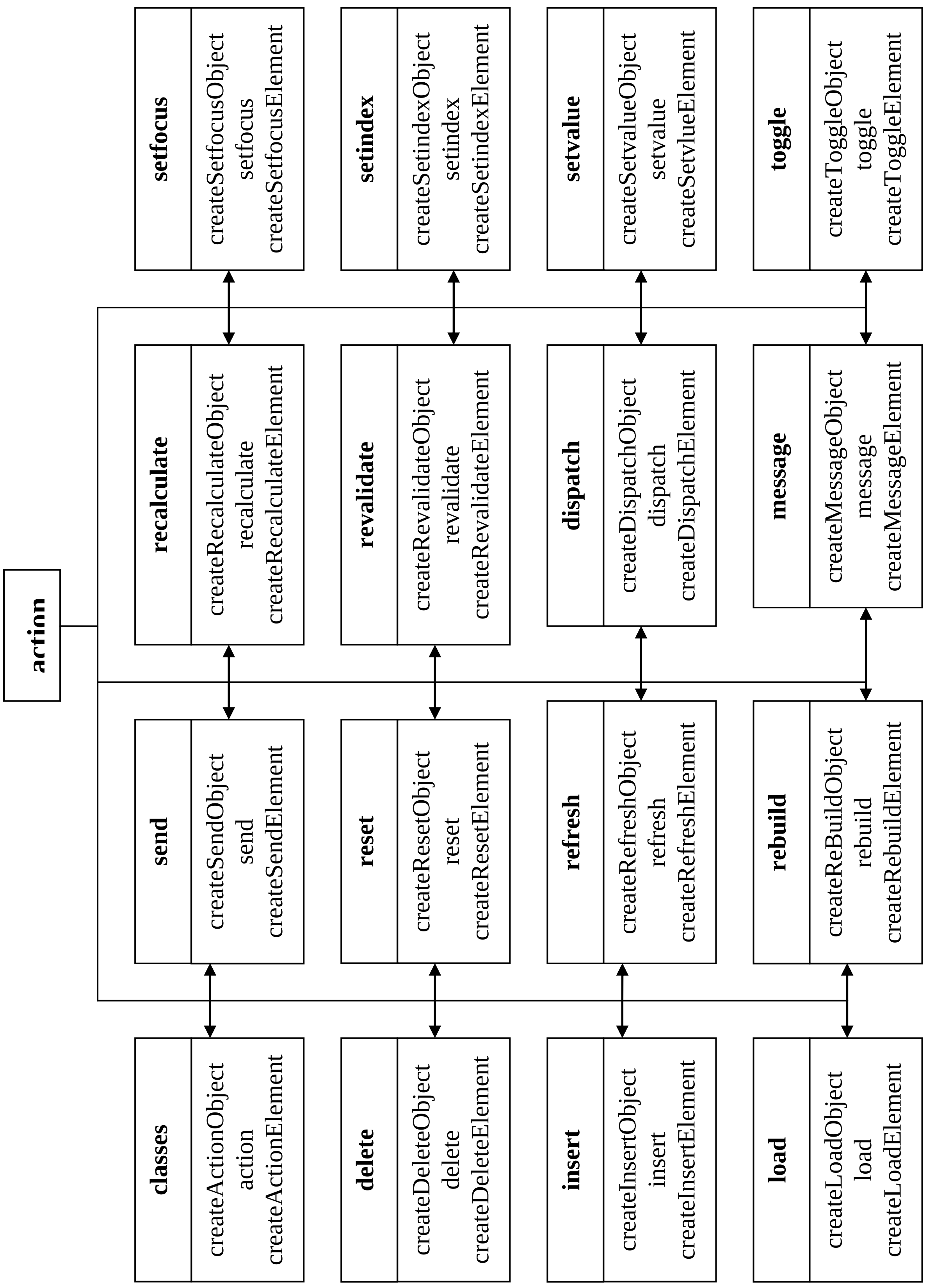


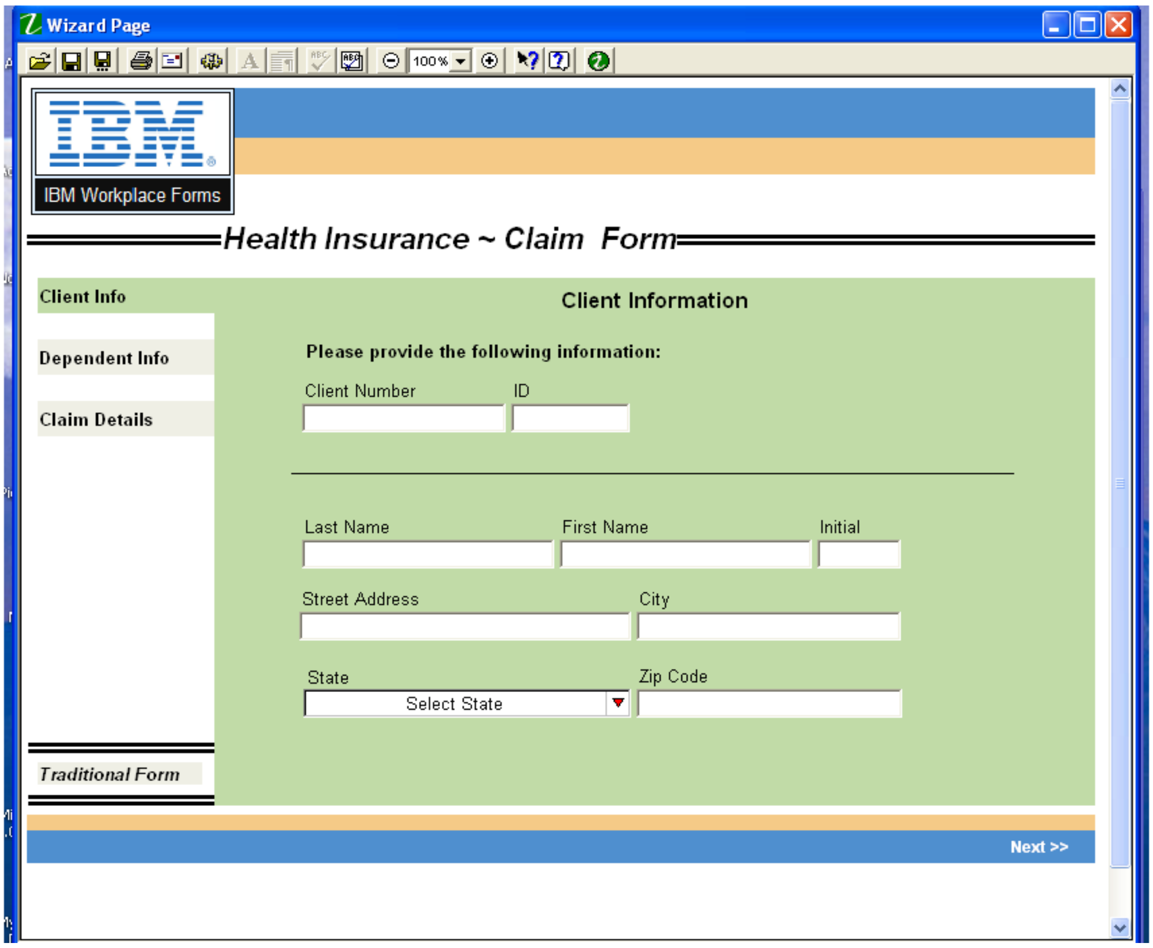
Fig. 5.11 The action Package

6. RESULTS

6.1 SAMPLE FORM (HealthDemoInsurance FORM)

The form is taken from the samples forms provided with the IBM Viewer v2.6. It consists of 2 pages shown below. This is a very complicated form and has been simplified using our design.The original code was about 48 pages in length and the modified code is 28 pages, a **reduction of about 42%.** We present here, only the code for page 1(case 1) of the form as given below.

PAGE 1 (View 1)



Health Insurance ~ Claim Form

Client Information

Please provide the following information:

Client Number
ID
Last Name
First Name
Initial
Street Address
City
State
Select State
Zip Code

XForms + XFDL Form

```
<?xml version="1.0" encoding="UTF-8"?>
<XFDL xmlns:custom=”http://www.ibm.com/xmlns/prod/XFDL/Custom”
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xfdl="http://www.ibm.com/xmlns/prod/XFDL/7.0"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns="http://www.ibm.com/xmlns/prod/XFDL/7.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<globalpage sid="global">
      <global sid="global">

         <xformsmodels>
            <xforms:model>
               <xforms:instance id="INSTANCE1" xmlns="">
                  <claimRequest>
                     <client>
                        <clientNum></clientNum>
                        <id></id>
                        <fname></fname>
                        <mname></mname>
                        <lname></lname>
                        <addressInfo>
                           <street></street>
                           <city></city>
                           <state></state>
                           <code></code>
                        </addressInfo>
                     </client>
                     <dependents>
                        <dependent>
                           <depID></depID>
                           <fname></fname>
                           <lname></lname>
                        </dependent>
                        <dependent>
                           <depID></depID>
                           <fname></fname>
                           <lname></lname>
                        </dependent>
                     </dependents>
                     <claim>
                        <drug>
                           <drugClaim>no</drugClaim>
                           <drugItem>
                              <id></id>
                              <date></date>
                              <amount>0</amount>
                           </drugItem>
                           <drugItem>
                              <id></id>
                              <date></date>
                              <amount>0</amount>
                           </drugItem>
                           <totalAmount></totalAmount>
                        </drug>
                        <services>
                           <serviceClaim>no</serviceClaim>
                           <serviceItem>
                              <id></id>
                              <date></date>
                              <doctor></doctor>
                              <amount>0</amount>
                           </serviceItem>
                           <serviceItem>
                              <id></id>
                              <date></date>
                              <doctor></doctor>
                              <amount>0</amount>
                           </serviceItem>
                           <totalAmount></totalAmount>
                        </services>
                        <totalClaim></totalClaim>
                     </claim>
                     <claimReceipt></claimReceipt>
                  </claimRequest>
               </xforms:instance>

               <xforms:instance id="prepop" xmlns="">
                  <states>
                        <state abbrev="AL">Alabama</state>
                        <state abbrev="AK">Alaska</state>
                        <state abbrev="AS">American Samoa</state>
                        <state abbrev="AA">APO FPO Americas</state>
                        <state abbrev="AE">APO FPO Can Euro</state>
                        <state abbrev="AP">APO FPO Pacific</state>
                        <state abbrev="AZ">Arizona</state>
                        <state abbrev="IA">Iowa</state>
                        <state abbrev="KS">Kansas</state>
                     </states>
               </xforms:instance>
               <xforms:instance id="calculations" xmlns="">
                  <data>
                     <dependentAddBtn></dependentAddBtn>
                     <drugAddBtn></drugAddBtn>
                     <servicesAddBtn></servicesAddBtn>
                  </data>
               </xforms:instance>

<xforms:bind nodeset="dependents/dependent[position()=last()]"
relevant="false()"> </xforms:bind>
<xforms:bind nodeset="claim/drug/drugItem[position()=last()]"
relevant="false()"> </xforms:bind>
<xforms:bind nodeset="claim/services/serviceItem[position()=last()]"
relevant="false()"> </xforms:bind>

<xforms:bind calculate="sum(instance('INSTANCE1')/claim/services/
serviceItem/amount)" nodeset="instance('INSTANCE1')/claim/services/
totalAmount"> </xforms:bind>
<xforms:bind calculate="sum(instance('INSTANCE1')/claim/drug/drugItem
/amount)" nodeset="instance('INSTANCE1')/claim/drug/totalAmount">
</xforms:bind>
<xforms:bind calculate="instance('INSTANCE1')/claim/drug/totalAmount
+ instance('INSTANCE1')/claim/services/totalAmount" nodeset="instance
('INSTANCE1')/claim/totalClaim"> </xforms:bind>

<xforms:bind id="dependentAddBtnRelevant" nodeset="instance
('calculations')/dependentAddBtn" relevant="(count(instance
('INSTANCE1')/dependents/dependent) &lt; 11)"> </xforms:bind>
```

```
<xforms:bind id="drugAddBtnRelevant"
nodeset="instance('calculations') /drugAddBtn"
relevant="(count(instance('INSTANCE1')/claim/drug/ drugItem) &lt;
9)"> </xforms:bind>
<xforms:bind id="serviceAddBtnRelevant" nodeset="instance
('calculations')/servicesAddBtn"
relevant="(count(instance('INSTANCE1') /claim/services/serviceItem)
&lt; 9)"> </xforms:bind>

<xforms:bind calculate="if(count(instance('INSTANCE1')/claim/drug
/drugItem) &lt; 3 and instance('INSTANCE1')/claim/drug/drugItem/id =
'', 'no', 'yes')" id="drugIdChange" nodeset="instance('INSTANCE1')/
claim/drug/drugClaim"> </xforms:bind>

<xforms:submission action="file:data.xml" id="submit1" instance=
"INSTANCE1" method="put" ref="/claimRequest" replace="none">
</xforms:submission>

</xforms:model>
</xformsmodels>
         <custom:selectedCase>CASE1</custom:selectedCase>
         <custom:selectedClaim>CASE12</custom:selectedClaim>
         <custom:mainColor>#C1DBA7</custom:mainColor>
         <custom:notFocusedColor>#EFEFE5</custom:notFocusedColor>
         <custom:claimCase>CASE12</custom:claimCase>

</global>
</globalpage>
<page sid="wizardPage1">
      <global sid="global">
         <label>Wizard Page</label>
      </global>

      <label sid="mainImage">
         <itemlocation>
            <x>0</x>
            <y>0</y>
            <width>156</width>
            <height>102</height>
         </itemlocation>
         <imagemode>resize</imagemode>
         <value></value>
         <justify>center</justify>
         <bgcolor>#FFFFFF</bgcolor>
         <fontinfo>
            <fontname>Arial</fontname>
            <size>12</size>
            <effect>bold</effect>
            <effect>italic</effect>
         </fontinfo>
         <image>PAGE1.Form_Branding_1</image>
      </label>
      <box sid="toolbarBox1">
         <itemlocation>
            <x>139</x>
            <y>3</y>
            <width>653</width>
            <height>40</height>
         </itemlocation>
         <border>off</border>
         <!-- changing this background color will change background
color on rest of form-->
         <bgcolor>#4F8FCF</bgcolor>
      </box>
      <box sid="toolbarBox2">
         <itemlocation>
            <x>139</x>
            <y>41</y>
            <width>653</width>
            <height>28</height>
         </itemlocation>
         <border>off</border>
         <!-- changing this background color will change background
color on rest of form-->
         <bgcolor>#F5CA87</bgcolor>
      </box>
      <box sid="toolbarBox3">
         <itemlocation>
            <x>0</x>
            <y>558</y>
            <width>792</width>
            <height>26</height>
         </itemlocation>
         <border>off</border>
         <bgcolor
compute="wizardPage1.toolbarBox2.bgcolor">#F5CA87</bgcolor>
      </box>
      <box sid="toolbarBox4">
         <itemlocation>
            <x>0</x>
            <y>570</y>
            <width>792</width>
            <height>25</height>
         </itemlocation>
         <border>off</border>
         <bgcolor
compute="wizardPage1.toolbarBox1.bgcolor">#4F8FCF</bgcolor>
      </box>

      <line sid="tradTopLine1">
         <itemlocation>
            <x>1</x>
            <y>503</y>
            <width>138</width>
            <height>3</height>
         </itemlocation>
         <thickness>3</thickness>
      </line>
      <line sid="tradTopLine2">
         <itemlocation>
            <x>1</x>
            <y>508</y>
            <width>138</width>
            <height>3</height>
         </itemlocation>
         <thickness>3</thickness>
      </line>
      <line sid="tradBottomLine1">
         <itemlocation>
            <x>1</x>
            <y>543</y>
            <width>138</width>
            <height>3</height>
         </itemlocation>
         <thickness>3</thickness>
      </line>
      <line sid="tradBottomLine2">
         <itemlocation>
            <x>1</x>
            <y>548</y>
            <width>138</width>
            <height>3</height>
         </itemlocation>
         <thickness>3</thickness>
      </line>
      <line sid="titleLine1">
         <itemlocation>
            <x>0</x>
            <y>115</y>
            <width>792</width>
            <height>3</height>
         </itemlocation>
         <thickness>3</thickness>
      </line>
      <line sid="titleLine2">
         <itemlocation>
            <x>0</x>
            <y>120</y>
            <width>792</width>
            <height>3</height>
         </itemlocation>
         <thickness>3</thickness>
      </line>

      <button sid="BUTTON1">
         <xforms:trigger>
            <xforms:label>Client Info</xforms:label>
            <xforms:action ev:event="DOMActivate">
               <xforms:toggle case="CASE1"></xforms:toggle>
            </xforms:action>
         </xforms:trigger>
         <itemlocation>
            <x>6</x>
            <y>146</y>
            <width>136</width>
            <height>32</height>
         </itemlocation>
            <bgcolor compute="global.global.custom:selectedCase ==
            'CASE1' ? (global.global.custom:mainColor) :
            (global.global.custom:notFocusedColor)">#EFEFE5</bgcolor>
         <border>off</border>
            <custom:onActivated xfdl:compute="toggle(wizardPage1.
            BUTTON1.activated, 'on', 'off') == '1'
            ? (set('global.global.custom:selectedCase', 'CASE1'))
            : ''"></custom:onActivated>
         <justify>left</justify>
         <fontinfo>
            <fontname>Arial</fontname>
            <size>8</size>
            <effect>bold</effect>
         </fontinfo>
         <previous>next</previous>
```

```
</button>
<button sid="BUTTON2">
   <xforms:trigger>
      <xforms:label>Dependent Info</xforms:label>
      <xforms:action ev:event="DOMActivate">
         <xforms:toggle case="CASE2"></xforms:toggle>
      </xforms:action>
   </xforms:trigger>
   <itemlocation>
      <x>6</x>
      <y>193</y>
      <width>136</width>
      <height>32</height>
   </itemlocation>
      <bgcolor compute="global.global.custom:selectedCase ==
      'CASE2' ? (global.global.custom:mainColor) :
      (global.global.custom:notFocusedColor)">#C1DBA7</bgcolor>
   <border>off</border>
      <custom:onActivated xfdl:compute="toggle(wizardPage1
      .BUTTON2.activated, 'on', 'off') == '1'
      ? (set('global.global.custom:selectedCase', 'CASE2'))
      : ''"></custom:onActivated>
   <justify>left</justify>
   <fontinfo>
      <fontname>Arial</fontname>
      <size>8</size>
      <effect>bold</effect>
   </fontinfo>
</button>
<button sid="BUTTON3">
   <xforms:trigger>
      <xforms:label>Claim Details</xforms:label>
      <xforms:action ev:event="DOMActivate">
         <xforms:toggle case="CASE3"></xforms:toggle>
      </xforms:action>
   </xforms:trigger>
   <itemlocation>
      <x>6</x>
      <y>240</y>
      <width>136</width>
      <height>32</height>
   </itemlocation>
      <bgcolor compute="global.global.custom:selectedCase ==
      'CASE3' ? (global.global.custom:mainColor) :
      (global.global.custom:notFocusedColor)">#C1DBA7</bgcolor>
   <border>off</border>
       <custom:onActivated xfdl:compute="toggle(wizardPage1
      .BUTTON3.activated, 'on', 'off') == '1'
      ? (set('global.global.custom:selectedCase', 'CASE3'))
      : ''"></custom:onActivated>
   <justify>left</justify>
   <fontinfo>
      <fontname>Arial</fontname>
      <size>8</size>
      <effect>bold</effect>
   </fontinfo>
   <next>menuTradForm</next>
</button>
<button sid="menuTradForm">
   <value>Traditional Form</value>
   <fontinfo>
```

```
      <fontname>Arial</fontname>
      <size>8</size>
      <effect>bold</effect>
      <effect>italic</effect>
   </fontinfo>
   <border>off</border>
   <fontcolor>#000000</fontcolor>
      <bgcolor
      compute="(global.global.custom:notFocusedColor)">
      #EFEFE5</bgcolor>
   <itemlocation>
      <x>6</x>
      <y>516</y>
      <width>127</width>
      <height>22</height>
   </itemlocation>
   <justify>left</justify>
   <type>pagedone</type>
   <url>#PAGE1.global</url>
</button>

<pane sid="PANE1">
   <xforms:switch ref="instance('INSTANCE1')">
      <xforms:case id="CASE1" selected="true">
         <label sid="LABEL1">
            <xforms:output>
               <xforms:label>Client Information</xforms:label>
            </xforms:output>
            <itemlocation>
               <x>253</x>
               <y>4</y>
            </itemlocation>
            <fontinfo>
               <fontname>Arial</fontname>
               <size>10</size>
               <effect>bold</effect>
            </fontinfo>
         </label>
         <field sid="FIELD6">
            <xforms:input ref="client/clientNum">
               <xforms:label></xforms:label>
            </xforms:input>
            <itemlocation>
               <x>64</x>
               <y>75</y>
               <width>151</width>
            </itemlocation>
            <scrollhoriz>wordwrap</scrollhoriz>
            <value></value>
            <label>Client Number</label>
         </field>
         <field sid="FIELD7">
            <xforms:input ref="client/id">
               <xforms:label></xforms:label>
            </xforms:input>
            <itemlocation>
               <x>219</x>
               <y>75</y>
               <width>88</width>
            </itemlocation>
            <scrollhoriz>wordwrap</scrollhoriz>
            <value></value>
            <label>ID</label>
         </field>

         <line sid="LINE1">
            <itemlocation>
               <x>56</x>
               <y>148</y>
               <width>536</width>
            </itemlocation>
         </line>
         <field sid="FIELD3">
            <xforms:input ref="client/lname">
               <xforms:label></xforms:label>
            </xforms:input>
            <itemlocation>
               <x>64</x>
               <y>179</y>
               <width>187</width>
            </itemlocation>
            <scrollhoriz>wordwrap</scrollhoriz>
            <label>Last Name</label>
         </field>
         <field sid="FIELD1">
            <xforms:input ref="client/fname">
               <xforms:label></xforms:label>
            </xforms:input>
            <itemlocation>
               <x>328</x>
               <y>147</y>
               <width>187</width>
               <after>FIELD3</after>
            </itemlocation>
            <scrollhoriz>wordwrap</scrollhoriz>
            <label>First Name</label>
         </field>
         <field sid="FIELD2">
            <xforms:input ref="client/mname">
               <xforms:label></xforms:label>
            </xforms:input>
            <itemlocation>
               <x>245</x>
               <y>50</y>
               <width>62</width>
               <after>FIELD1</after>
               <after>FIELD1</after>
            </itemlocation>
            <scrollhoriz>wordwrap</scrollhoriz>
            <label>Initial</label>
         </field>
         <field sid="FIELD4">
            <xforms:input ref="client/addressInfo/street">
               <xforms:label></xforms:label>
            </xforms:input>
            <itemlocation>
               <x>62</x>
               <y>234</y>
            </itemlocation>
            <scrollhoriz>wordwrap</scrollhoriz>
            <label>Street Address</label>
         </field>
```

```
<field sid="FIELD5">
   <xforms:input ref="client/addressInfo/city">
      <xforms:label></xforms:label>
   </xforms:input>
   <itemlocation>
      <x>312</x>
      <y>234</y>
      <width>196</width>
   </itemlocation>
   <scrollhoriz>wordwrap</scrollhoriz>
   <label>City</label>
</field>

<label sid="stateUSLabel1">
   <itemlocation>
      <x>65</x>
      <y>293</y>
   </itemlocation>
   <value>State</value>
</label>

<popup sid="stateUS1">
   <itemlocation>
      <x>65</x>
      <y>313</y>
      <width>243</width>
   </itemlocation>
   <size>
      <width>20</width>
      <height>1</height>
   </size>
   <label>Select State</label>

   <xforms:select1 ref="client/addressInfo/state">
      <xforms:label>Select State</xforms:label>
      <xforms:itemset nodeset="instance('prepop')
   /state">
         <xforms:label></xforms:label>
         <xforms:value ref="@abbrev"></xforms:value>
      </xforms:itemset>
   </xforms:select1>

</popup>

<field sid="zipCode1">
   <format>
      <datatype>string</datatype>
      <constraints>
         <patterns>
            <pattern>(\d{5})</pattern>
            <pattern>(\d{5})-?(\d{4})</pattern>
         </patterns>
      </constraints>
      <presentation>
         <patternrefs>
            <patternref>$1</patternref>
            <patternref>$1-$2</patternref>
         </patternrefs>
         <casetype>upper</casetype>
      </presentation>
   </format>
```

```
            <value></value>
            <itemlocation>
               <x>312</x>
               <y>293</y>
               <width>197</width>
            </itemlocation>
            <xforms:input ref="client/addressInfo/code">
               <xforms:label></xforms:label>
                  <xforms:help>Enter the ZIP Code in either
                  ##### or #####-#### format</xforms:help>
            </xforms:input>
            <label>Zip Code</label>
         </field>

         <label sid="LABEL2">
            <itemlocation>
               <x>64</x>
               <y>44</y>
            </itemlocation>
            <value>Please provide the following
            information:</value>
            <fontinfo>
               <fontname>Arial</fontname>
               <size>8</size>
               <effect>bold</effect>
            </fontinfo>
         </label>

         <spacer sid="SPACER1">
            <itemlocation>
               <x>637</x>
               <y>379</y>
            </itemlocation>
         </spacer>
      </xforms:case>
```

New Code

```
<xforms>
      <model>
            <instance id="INSTANCE1">
                  <claimRequest>
                        <client>
                              <clientNum></clientNum>
                              <id></id>
                              <fname></fname>
                              <mname></mname>
                              <lname></lname>
                              <addressInfo>
                                    <street></street>
                                    <city></city>
                                    <state></state>
                                    <code></code>
                        </addressInfo>
                        </client>

                        <dependents>
                        <dependent>
```

```
                <depID></depID>
                <fname></fname>
                <lname></lname>
        </dependent>
        <dependent>
                <depID></depID>
                <fname></fname>
                <lname></lname>
        </dependent>
        </dependents>

        <claim>
            <drug>
                <drugClaim>no</drugClaim>
                <drugItem>
                    <id></id>
                    <date></date>
                    <amount>0</amount>
                </drugItem>
                <drugItem>
                    <id></id>
                    <date></date>
                    <amount>0</amount>
                </drugItem>
                <totalAmount></totalAmount>
            </drug>

            <services>
                <serviceClaim>no</serviceClaim>
                <serviceItem>
                    <id></id>
                    <date></date>
                    <doctor></doctor>
                    <amount>0</amount>
                </serviceItem>
                <serviceItem>
                    <id></id>
                    <date></date>
                    <doctor></doctor>
                    <amount>0</amount>
                </serviceItem>
                <totalAmount></totalAmount>
            </services>
        <totalClaim></totalClaim>
        </claim>
        <claimReceipt></claimReceipt>
    </claimRequest>
</instance>

<instance id="prepop">
    <states>
    <state abbrev="AL">Alabama</state>
    <state abbrev="AK">Alaska</state>
    <state abbrev="AS">American Samoa</state>
    <state abbrev="AA">APO FPO Americas</state>
    <state abbrev="AE">APO FPO Can Euro</state>
    <state abbrev="AP">APO FPO Pacific</state>
    <state abbrev="AZ">Arizona</state>
    <state abbrev="IA">Iowa</state>
    <state abbrev="KS">Kansas</state>
```

```
                </states>
        </instance>

        <instance id="calculations">
                <data>
                        <dependentAddBtn></dependentAddBtn>
                        <drugAddBtn></drugAddBtn>
                        <servicesAddBtn></servicesAddBtn>
                </data>
        </instance>

        <submission id="submit1"
                     how="put"
                     where="file:data.xml
                     instance="INSTANCE1"/>
</model>

<control>

        <calc path="instance('INSTANCE1')/claim">
                <var>/services/totalAmount, /drug/totalAmount, /totalAmount</var>
                <expr>sum(/services/serviceItem/amount),
                      sum(/drug/drugItem/amount),
                      services/totalAmount + drug/totalAmount
                </expr>
        </calc>

        <valid>
                <var>dependents/dependent[position() = last()],
                     claim/drug/drugItem[position() = last()],
                    claim/services/serviceItem[position() = last()]
                </var>
                <relevant>false(), false(), false()</relevant>
        </valid>

        <valid path = "instance('calculations')">
                <var>/dependentAddBtn, /drugAddBtn, /servicesAddBtn</var>
                        <relevant>(count(instance('INSTANCE1')/dependent) &lt; 11,
                        (count(instance('INSTANCE1')/claim/drug/drugItem) &lt; 9,
                        (count(instance('INSTANCE1')/claim/services/serviceItem)
                        &lt; 9 </relevant>
        </valid>
</control>

<view>
        <global>

                <custom:selectedCase>CASE1</custom:selectedCase>
                <custom:selectedClaim>CASE12</custom:selectedClaim>
                <custom:mainColor>#C1DBA7</custom:mainColor>
                <custom:notFocusedColor>#EFEFE5</custom:notFocusedColor>
                <custom:claimCase>CASE12</custom:claimCase>

                <fontinfo id="fnt">
                        <fontname>Arial</fontname>
                        <size>8<size>
                        <effect>italic</effect>
                </fontinfo>

                <fontinfo id="fnt1" use="fnt">
```

```
                        <effect>bold</effect>
                <**/fontinfo**>

        **</global>**
        **<page id="wizardPage1" label="Wizard Page">**
        **<global>**
                **<fontinfo id="fnt2">**
                <fontname>Helvetica</fontname>
                        <effect>bold</effect>
                **</fontinfo>**
        **</global>**

        <image id="mainImage">
                <filename>PAGE1.Form_Branding_1</filename>
                <imagemode>resize</imagemode>
                <justify>center</justify>
                <fontinfo use="fnt">
                        <size>12</size>
                </fontinfo>
        </image>

         <box border="off">
                <id>toolbarBox1,toolbarBox2,toolbarBox3,toolbarBox4</id>
         </box>

         <line thickness="3">
                <id>tradTopLine1,tradTopLine2,tradBottomLine1,tradBottomLine2,titl
                eLine1, titleLine2</id>
        </line>

        <button border="off">
                 <fontinfo use="fnt1">
                </fontinfo>
                <trigger>
                        <id>BUTTON1,BUTTON2,BUTTON3</id>
                        <label>Client Info,Dependent Info,Claim Details</label>
                <action event="DOMActivate">
                                <toggle case="CASE1,CASE2,CASE3"></toggle>
                         </action>
                 </trigger>
        </button>

    <bgcolor compute="custom:selectedCase == 'CASE1' ? custom:mainColor :
custom:notFocusedColor">#EFEFE5</bgcolor>

    <custom:onActivated compute="toggle(BUTTON1.activated, 'on', 'off') == '1' ?
(set('custom:selectedCase', 'CASE1')) : ""></custom:onActivated>

     <bgcolor compute="custom:selectedCase == 'CASE2' ? custom:mainColor :
custom:notFocusedColor">#C1DBA7</bgcolor>

     <custom:onActivated compute="toggle(BUTTON2.activated, 'on', 'off') == '1'?
(set('custom:selectedCase', 'CASE2')): ""></custom:onActivated>

     <bgcolor compute="custom:selectedCase == 'CASE3' ? custom:mainColor :
custom:notFocusedColor">#C1DBA7</bgcolor>

     <custom:onActivated compute="toggle(BUTTON3.activated, 'on', 'off') == '1'?
(set('custom:selectedCase', 'CASE3')): ""></custom:onActivated>
```

```
  <button id="menuTradForm">
  <value>Traditional Form</value>
  <fontinfo use="fnt">
  </fontinfo>
  <border>off</border>
  <fontcolor>#000000</fontcolor>
  <bgcolor>#EFEFE5</bgcolor>

  <type>pagedone</type>
  <url>global</url>
</button>

<pane id="PANE1">
  <switch what="instance('INSTANCE1')">

    <case id="CASE1" selected="true">

       <output id= "LABEL1">
         <label>Client Information</label>
         <fontinfo use="fnt1">
            <size>10</size>
         </fontinfo>
       </output>

       <input path="client" scrollhoriz = "wordwrap">
            <id>FIELD6,FIELD7,FIELD3,FIELD1,FIELD2</id>
            <what>/clientNum, /id, lname, fname, mname</what>
    <label>Client Number, ID, Last Name, First Name, Initial</label>
       </input>

        <input path="client/addressInfo" scrollhoriz = "wordwrap">
            <id>FIELD4,FIELD5</id>
            <what> /street, /city</what>
            <label>Street Address, City</label>
        </input>

       <line id="LINE1"></line>

       <label id="stateUSLabel1">
            <value>State</value>
       </label>

       <select type="one" id="stateUS1" what="client/addressInfo/state">
         <label>Select State</label>
         <items what="instance('prepop')/state">
           <value what="@abbrev"></value>
         </items>
       </select>

       <input what="client/addressInfo/code" id="zipCode1">
         <help>Enter the ZIP Code in either ##### or #####-#### format</help>

          <format>
         <datatype>string</datatype>

         <constraints>
           <patterns>
             <pattern>(\d{5})</pattern>
             <pattern>(\d{5})-?(\d{4})</pattern>
           </patterns>
```

```
      </constraints>

      <presentation>
        <patternrefs>
          <patternref>$1</patternref>
          <patternref>$1-$2</patternref>
        </patternrefs>
        <casetype>upper</casetype>
      </presentation>

    </format>
     <label>Zip Code</label>
    </input>

  <label id="LABEL2">
    <value>Please provide the following information:</value>
    <fontinfo use="fnt1"/>
  </label>

  <spacer id="SPACER1"> </spacer>
</case>
```

7. CONCLUSION AND FUTURE WORK

XForms is an open standard developed in order to eliminate the need for scripting, provide device independence etc. As it is an XML standard, it fits well into the XML workflows. XForms is build on the lines of MVC architecture. The view part specifies the purpose of the form control rather than how it is displayed. So, XForms requires a Presentation language for display.

XFDL is the best presentation option currently available for XForms. XFA is another new generation presentation language, but it doesnot support XForms. In this thesis, certain important features and functions of XFA have been integrated into XFDL. Also, some of XFDL and XForms tags have been simplified. A converter is designed that converts this new XFDL code to the original. In the process, we have also simplified some of XForms tags.

Separate layout tag is provided for specifying the layout information. It helps reduce the size of the code by a large amount. Because in XFDL code, one finds the itemlocation tag (specifying the coordinates and size of items), scattered all over the form. Also, a use attribute is provided for specifying the common information at one place and reusing it again and again. This corresponds to the prototype tag of XFA.

As future work, instead of converting the modified code to original XFDL code, an engine can be designed that directly displays the form written using our code. XHTML is another presentation language that must be improved on the lines of XFDL and further based on our work. CSS (Cascading Stylesheets) is a language for styling HTML and other forms. An XML version of this will ensure its better use with XML based languages.

APPENDIX A

Mapping Tables

Following are the tables mapping the modified language **with XForms + XFDL languages :**

General Features

Purpose	XForms + XFDL	Our Language
XML Based	Yes	Yes
Root Element	<XFDL>	<xforms>
Default Namespaces	No, Namespaces must be explicitly defined <XFDL xmlns= "http://www.ibm.com/xmlns/prod/XFDL/7.0" xmlns:xforms= "http://www.w3.org/2002/xforms" xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom" xmlns:ev="http://www.w3.org/2001/xml-events" xmlns:xsd= "http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">	Yes, these namespaces are defined by default.
Other Namespaces	<XFDL xmlns:prefix=”URI” ……….>	<xforms xmlns:prefix=”URI”>
globalpage	<globalpage sid="global">	XXX
Global options	<global sid="global">	<global>
Identfier	sid (scope identifier)	id (identifier)
Page definition	<page>	<page>
Seperation of common information	XXX	<global> <common tag id=”id”> common info

reused using **use**		</common tag> </global> <common tag use=”id”/>
actions	Xforms Actions	Same
functions	XForms + XPath + XFDL Functions	Some additional functions namely concat, str, max, min, atan2, avg, count, and some financial functions
events	XML Events	Same
Custom options	Allowed	Allowed, namespace for custom items is by default “custom”

Model and Control

Purpose	**XForms + XFDL**	**Our Language**
XForms Model	<xformsmodels> (All models within this tag)	XXX
To specify a data model	<xforms: model id=”modelId” schema =”filename” ………> <xforms:instance src=”filename”/> Or <xforms:instance> ….Instance data… </xforms:instance> <xforms:schema> Schema </xforms:schema> . …Control part … </xforms:model>	<model id=”modelId” schema =”filename”> <data>filename Or ….Data…. </data> <schema> filename or xml schema </schema> </model>
To specify instance	<xforms:instance>	<data>
control	<xforms:model ….. events=” ” function=”QName”> <xforms:bind nodeset=”Path” calculate or property …. > </xforms:bind> <xforms:submission …. /> <xforms:action> …….. </xforms:action> </xforms:model>	<control events=” “ function=”QName”> <calc> calculations </calc> <valid> validations </valid> <submission ….. /> <action> …….. </action>

calculations	<xforms:bind nodeset=”Path” calculate=”expr”/>
	<xforms:bind 2 ….
	Separate binds <xforms:bind>
		<calc>
		    <var> var1,var2 … </var>
		    <expr> expr1,expr2...
		    </expr>
		</calc>

validations	<xforms:bind nodeset=”Path” relevant or another property …. =”condition”/>
	<xforms:bind 2 ….
	Separate binds <xforms:bind>
		<valid>
		    <var> var1,var2 … </var>
		    <any property ....> expr1,expr2...
		    </any property>
		</valid>

submission	<xforms:submission id=”id” action= “file or URI” ref=”what to submit “ method= “http method (get|put|post)” ……. />	<submission id=”id” where= ”file or URI” what=” ” how=”method” ………. />
Submission of encrypted data	No, data submitted as xml is visible to all	Yes, <submission id=”id” encryption= ”encryption algo” ………. />

View (Form Controls)

Purpose	XForms + XFDL	Our Language
Input (field)	<field sid=”id”> <xforms:input ref=”Path” ……. > <xforms:label>label </xforms:label> help|hint|alert|action </xforms:input> </field>	<input id=”id” what=”Path” ……. help|hint|alert|action|label > </input>

More than one input	<field sid=”id1”>
	<xforms:input ref=”Path” ……. >
	    <xforms:label>label
	    </xforms:label>
	    help|hint|alert|action
	
	</xforms:input>
	</field>
	<field sid=”id2” ….. same as above>
	</field>
		<input ……. >
		    <id> id1,id2 </id>
		    <what> Path1 </what>
		    <help|hint|alert|action|label>
		</input>

Default Enclosure	No, eg- field must always skin input	Yes

secret	<field sid=”id”> <xforms:secret ref=”Path” …… > help|hint|alert|action|label </xforms:secret> </field>	<input type=”secret” id=”id” what=”Path” ……. help|hint|alert|action|label > </input>
textarea	<field sid=”id”> <xforms:textarea ref=”Path” …… > …… </xforms:textarea> </field>	<input type=”textarea” id=”id” what=”Path” … > ……… </input>
output	<label sid=”id”> <xforms:output ref=”Path” …… > …… </xforms:output> </label>	<output id=”id” what=”Path” … > ……… </output>
select	<checkgroup sid=”id”> <xforms:select ref=”Path” …> <xforms:label>… </xforms:label> <xforms:item>... <xforms:label>… </xforms:label> <xforms:value>… <xforms:value> </xforms:item> <xforms:item>... <xforms:label>… </xforms:label> <xforms:value>… <xforms:value> </xforms:item> <xforms:select> </checkgroup>	<select displayType=”checkgroup” id=”id” what=”Path” > <label>… </label> <items> <label>label1,2,… </label> <value>value1,2… </value> </items> </select>
select1	<select1 …>	<select type=”one” …>
Set of items within the select clause	Itemset	items
Table heading	As separate labels before the table tag	Labels within the elements of the table
Insert/delete buttons	As separate buttons outside the table, have to be explicitly linked to the table through the ref attribute	Part of the table itself. Advantage : need not be linked to the table <trigger> <id>but1,but2</id> <type>add,delete</type>

		<label>……</label> </trigger>
Seperation of layout information	No, coordinate information present in each item	Yes, separate layout for each page

Location and size of items	<itemlocation>
	    <x> x-coordinate </x>
	    <y> y-coordinate </y>
	    <width> ….. </width>
	    <height> …. </height>
	        ………….
	</itemlocation> within each item
		<layout>
		    <items>
		        <id> id1, id2 ….</id>
		        <coord>(x1,y1), (x2,y2) , ...
		        </coord>
		        <width> … , … </width>
		        <height> … , …</height>
		    </items>
		        …………
		</layout> at the end of page

coordinates	<x> x -coord </x> <y> y-coord </y>	<coord> (x,y) </coord>

REFERENCES

XML

[1] Aaron Skonnard, Martin Gudgin ,"Essential XML Quick Reference, A Programmer's Reference to XML, XPath,XSLT, XML Schema", Addison – Wesley Publication

[2] Tim Bray, October, 2000, "Extensible Markup Language (XML) 1.0 (Second Edition)", available at http://www.w3.org/TR/REC-xml .

[3] Tim Bray 1998, 2000, "Namespaces in XML", available at http://www.w3.org/TR/REC-xml/-names.

[4] developerWorks, August 2002, "Introduction to XML".

[5] Erik T. Ray ,"Learning XML, 2nd Edition", published by O'Reilly.

[6] Elliotte Rusty Harold and W. Scott Means*, "XML in a Nutshell, 2nd Edition",* published by O'Reilly.

XForms

[7] Micah Dubinko, "O'Reilly XForms Essentials"

[8] "XForms1.0, W3C Recommendation,14Oct,2003 ",available at http://www.w3.org/TR/2003/REC-xforms-20031014/

[9] Steven Pemberton, "XForms for HTML Authors", W3C Submission, 28 October 2003, available at www.w3.org/MarkUp/Forms/2003/xforms-for-html-authors.html

[10] Steven Pemberton, "XForms for HTML Authors", W3C Submission,2006-08-08, available at http://www.w3.org/MarkUp/Forms/2006/xforms-for-html-authors-part2.html

[11] XForms - The Next Generation of Web Forms, W3C Submission, available at www.w3.org/MarkUp/Forms/

[12] Richard Cardone, Danny Soroker, Alpana Tiwari, "Using XForms to Simplify Web Programming."

[13] Steven Pemberton, "XForms Quick Reference", W3C Submission, available at www.w3.org/MarkUp/Forms/2006/xforms-qr.html

XFDL

[14] “XFDL Specification”, IBM Workplace Forms, version 2.7.

[15] J. Boyer, T. Bray, & M. Gordon, “Extensible Forms Description Language (XFDL) 4.0”, W3C Note, available at: http://www.w3.org/TR/NOTE-XFDL

[16] Barclay T. Blair and John Boyer , “XFDL: Creating Electronic Commerce Transaction Records Using XML”

[17] John Boyer, “Enterprise-level Web Form Applications with XForms and XFDL”, IBM Corporation, November 2005

XFA

[18] XML Forms Architecture (XFA) Specification Version 2.4

[19] “XFA-Template, Version 1.0”, available at http://www.w3.org/1999/05/XFA/xfa-template

[20] Mike Tardif (JetForm), “XFA-FormCalc, Version 1.0”, available at http://www.w3.org/1999/05/XFA/xfa-formcalc.html

Parser

[21] Eric Armstrong, “Working with XML - The Java API for Xml Parsing (JAXP) Tutorial”, [Version 1.1, Update 31 -- 21 Aug 2001]

[22] Xerces Parser, available at http://xerces.apache.org

[23] developerWorks, July 2003, "Understanding DOM".

[24] Brett McLaughlin , “Java and XML, 2nd Edition”.

[25] Doug Tidwell , “XML programming in Java technology, Part 1,2,3”.

[26] LeHors, Arnaud, “Document Object Model (DOM) Level 2 Core Specification”, available at http://www.w3.org/TR/DOM-Level-2-Core/ ,1999