DEPOSIT-CONTRACT ON XML SCHEMA USER INTERFACE (XSI)

A Project Report Submitted in Partial Fulfillment of the Requirements

for the award of the Degree of

Master of Technology

in

Information Technology

(With Specialization in Banking Technology and Information Security)

By

R. Kabaleeshwaran

(08MCMB11)

Under the Guidance of

Dr. V.Radha

Institute for Development and Research in Banking Technology

Road No. 1, Castle Hills, Masab Tank,

Hyderabad – 500 057.

Submitted to

Department of Computer and Information Sciences

School of Mathematics and Computer/Information Sciences

University of Hyderabad

Hyderabad – 500 046

A.P., India.

April 2010

CERTIFICATE

This is to certify that the project work entitled "**Deposit-contract on XML Schema based universal User Interface (XSI)**", submitted for partial fulfillment of the requirements for the award of degree of **Master of Technology** in Information Technology (**with specialization in Banking Technology and Information Security**) to the **University of Hyderabad** is a bonafide project work carried out by **Mr. R. Kabaleeshwaran** (**Reg. No. 08MCMB11**) at **IDRBT** (Institute for Development and Research in Banking Technology), Hyderabad, for a period of one year during July 16,2009 to April 29,2010 under my guidance.

The core matter embodied in the project report has not been submitted for the award of any other degree or diploma.

Dr V.Radha
E-mail: vradha@idrbt.ac.in
Assistant Professor
IDRBT
Castle Hills, Masab Tank
Hyderabad – 500 057

CERTIFICATE

This is to certify that the project work entitled "**Deposit-contract on XML Schema based universal User Interface (XSI)**", submitted for partial fulfillment of the requirements for the award of degree of **Master of Technology in Information Technology** (M.Tech (I.T.)) (**with specialization in Banking Technology and Information Security**) to the **University of Hyderabad** is a record of bonafide project work carried out by **Mr. R. Kabaleeshwaran. (Reg.No. 08MCMB11)** at **IDRBT** (Institute for Development and Research in Banking Technology), Hyderabad, for a period of one year during July 16,2009 to April 30,2010 under the guidance of **Dr. V.Radha** (IDRBT).

Dr. V.Ravi,
Assistant Professor,
Coordinator – M.Tech (I.T.)
IDRBT,
Hyderabad.

Prof. Arun Agarwal,
Head of the Department, DCIS,
University of Hyderabad,
Gachibowli,
Hyderabad.

Prof. T. Amarnadh,
Dean (School of MCIS),
University of Hyderabad,
Gachi Bowli,
Hyderabad.

ACKNOWLEDGEMENTS

This project would not have been possible without the unstinting guidance and motivation of **Dr. V. Radha**, faculty, IDRBT. She gave me constant support through the thin and lean phases of project. I wish to express my sincere gratitude to her.

I thank **Mr. B. Sambamurthy**, Director, IDRBT, **Prof.Arun Agarwal** Head of Department, DCIS, University of Hyderabad and **Dr. Mahill Carr,** M. Tech Coordinator, IDRBT for all the infrastructure and technical support provided.

The guidance of all the faculty members of IDRBT and the Computer Sciences Department, University of Hyderabad has been precious and timely. I wish to thank them all for being very patient, understanding and helpful.

I would like to thank my super senior Mr. K. Pavan Kumar and Research Associate Mr. J. Anil Kumar for their kind help and uninterrupted support towards completion of my project.

R.Kabaleeshwaran

Kabaleesh86@gmail.com

University of Hyderabad and IDRBT

April 2010

RESEARCH PUBLICATION

Dr.V.Radha, R.Kabaleeshwaran, P.Chaitanya, "**Empowering Customers**" accepted and presented in International Conference on "**Emergent Business Models and Strategies for Knowledge Economy**" held at Indian Business Academy (IBA), Bangalore, Nov 19-21, 2009.

ABSTRACT

At present in India, each bank has got its own way of calculating interests on the deposits. In the context of floating interest rates, the customer is not in a position to assess the value of his deposit. The way the banks deal with the customer looks one sided, though the terms and conditions are explained in the back of the contract paper in a cryptic fashion. A business model that ensures transparency and fairness in conducting business helps both parties increase their efficiency and survival in the long run, which the present model lacks. We presented a model, where in the financial products like fixed deposit can be described in an XML schema, based on which any party can assess the value of product online. We present the model using the web services technologies like XSD, XML, SOAP, UDDI and XSI (XML Schema Interface) an in-house developed XML user interface.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Project Motivation

In current scenario, many organizations are receiving various kinds of forms from their clients by several means such as E-mail, documents. It is a cumbersome process for organizations to extract, store and process the information contained in these forms. Human intervention is required to handle these documents and to provide requested services. It is a time consuming process

In Banks one of the most popular financial products is Deposit. Deposit is a contract between the Customer and the Bank. Customer fixes his/her money in a bank for specific period of time as mentioned by the bank, Bank will pay the interest for that money and return the money (initial amount deposited plus interest accrued on that amount) on the maturity date or Customer has a provision to take money by canceling the Deposit contract as and when he requires. There are two types of Interest rates, one is fixed and the other is float. Fixed interest means constant rate of interest from issue to maturity date. Float interest rate keeps varying based on financial system parameters like Prime Lending Rate (PLR). Simple examples only have been taken for the purpose of explaining the model. However, the model can be extended further for complex products as well.

We propose to describe the Deposit contract using XML. The XML Schema specification standardized by the W3C serves the purpose of modeling XML data. It describes the structure and defines constraints of XML documents and becomes the key element in the exchange of XML based business data. It helps ensure data integrity and consistency at all levels of business application.

1.2 Objective

- To implement Banking & Financial products like Deposit, Loan contract etc., in XSI (XML Schema Interface) using XML and XML Schema.
- Provide structured communication between banks and customers through XML Schema.

- To replace the use of traditional browsers in all the financial transactions with the XSI based secure browser.
- To reduce the application development and maintenance.
- To minimize the dependence of business personnel on application developers.

1.3 Extensible Markup Language (XML)

1.3.1 Introduction

Markup encodes a description of the document's storage layout and logical structure. It is a way of conveying metadata. A markup language is a mechanism to identify structures in a document. XML specification [4] defines a standard way to add markup to documents. XML markup describes the semantics of content in a structured way. It separates presentation from semantics of content of a document.

XML was developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (W3C) in 1996. XML is defined as an application profile of SGML. SGML is the Standard Generalized Markup Language defined by ISO 8879. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade, but it is not well suited to serving documents over the web.

1.3.2 Why XML?

In HTML, both the tag semantics and the tag set are fixed. XML specifies neither tag semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships among them. XML was created so that richly structured documents could be used over the web.

1.3.3 Nature of XML

XML documents are composed of markup and content. There are six kinds of markup that can occur in an XML document: elements, entity references, comments, processing instructions, marked sections, and document type declarations which are used to describe data. XML is a text-based format, thus allowing it to exchange information across various platforms and devices. It is in a format that can be easily read by humans and machines.

1.3.4 Well-formed XML documents

A document is well formed if and only if it satisfies the syntax rules specified in XML specification [4]. Some of the syntax rules include:

- The document instance must conform to the grammar of XML documents,
- XML documents must contain at least one element
- Tags in XML are case sensitive
- There is a single 'root' element which contains all the other elements.
- Attribute values must always be quoted
- The beginning and end tags must match exactly
- Non-empty tags must be properly nested,
- No attribute may appear more than once on the same start-tag.

If XML document is well formed, then only XML processors will process that document.

1.3.5 Valid Documents

A well-formed document is valid only if it contains a proper document type declaration or schema and if the document obeys the constraints of that declaration (element sequence and nesting is valid, required attributes are provided, attribute values are of the correct type, namespaces are properly defined etc.). A validating processor can automatically identify the schema or DTD declaration from the XML document and then validate using it.

1.3.6 Document Object Model (DOM)

The Document Object Model (DOM) [12] is a standard that defines a platform- and language-neutral interface for accessing and manipulating the structure, data and style (CSS properties) of XML (and HTML) documents programmatically. The DOM provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. DOM is a tree-based API.

DOM compiles an XML document and then creates an in memory tree representation of the document as a node tree. When DOM parses the XML document, it breaks the document into

individual elements, attributes, and so on. It then allows applications to navigate and manipulate the contents of the document through the node tree using the DOM interface.

1.3.7 Simple API for XML (SAX)

The "Simple API for XML" (SAX) [7] is a standard interface that provides an event-driven, serial-access mechanism for element-by-element processing of XML document. SAX is an event-based API, developed collaboratively by the members of the XML-DEV mailing list, currently hosted by OASIS.

When a SAX parser reads the XML document as a stream of XML tags, it reports parsing events such as the start and end of elements, text sections, etc corresponding to various components of an XML document directly to the application through callbacks. The application implements handlers to deal with the different events much like handling events in a graphical user interface (GUI). Event driven approach is useful for large documents in which the program only needs to process a small portion of the document.

1.3.8 Difference between DOM and SAX

1. DOM reads the entire XML document into memory and stores it as a tree data structure where as SAX reads the XML document element by element and sends an event for each element that it encounters.
2. DOM provides "random access" into the XML document; SAX provides only sequential access to the XML document.
3. SAX is fast in terms of processing and requires very little memory, so it can be used for huge documents (or large numbers of documents).
4. Some DOM implementations have methods for manipulating data in the XML document in memory; SAX implementations do not.
5. SAX parsers generally require you to write a bit more code than the DOM interface.
6. Limited API in SAX -- every element is processed through the same event handler. You need to keep track of location of each element in document and in cases store temporary data.
7. DOM is slow and requires huge amounts of memory, so it cannot be used for large XML documents. Tree approach is useful for small documents in which the application needs to process a large portion of the document.
8. Unless you build a DOM style tree from application's internal representation for the data, it is not easy to write the XML document back to disk.

1.4 XML Schema Definition (XSD)

1.4.1 Introduction

W3C XML Schema Definition Language [8] is an XML language for describing and constraining the content of XML documents. XML Schemas provide mechanisms to define and describe the structure, content, and to some extent semantics of XML documents.

The purpose of a schema is to define a class of XML documents, and the term "instance document" is often used to describe an XML document that conforms to a particular schema. A schema document consists of a *schema* element as root element and a variety of sub-elements, most notably *element*, *complexType*, and *simpleType* which determine the appearance of elements and their content in instance documents.

Elements that contain sub-elements or carry attributes are said to have complex types, whereas elements that contain numbers (and strings, and dates, etc.) but do not contain any sub-elements are said to have simple types. Some elements have attributes; attributes always have simple types.

1.4.2 Namespaces

Each of the elements in the schema has a prefix say ‘xsd:’ or ‘xs:’ which is associated with the XML Schema namespace through the declaration, xmlns:xsd="http://www.w3.org/2001/XMLSchema", that appears as attribute in the *schema* element. The purpose of the association is to identify the elements and simple types as belonging to the vocabulary of the XML Schema language rather than the vocabulary of the schema author.

1.4.3 ComplexType Definitions

Named complex types are defined using the *complexType* element and such definitions typically contain a set of element declarations, element references, and attribute declarations. All attribute declarations must reference simple types because, unlike element declarations, attributes cannot contain other elements or other attributes. Sometimes it is preferable to use an existing element rather than declare a new element, In such cases, the value of the *ref* attribute must reference a global element, i.e. one that has been declared under *schema* rather than as part of a complex type definition. Any element which is declared with defined complex type in its *type*

attribute must appear with all those elements and attributes of complex type definition in the instance document.

1.4.4 Anonymous Type Definitions

Schemas can be constructed by defining sets of named types and then declaring elements that reference the types using the *type* construction. This style of schema construction is straightforward but it can be unwieldy, especially when many defined types are referenced only once and contain very few constraints. In these cases, a type can be succinctly defined as an anonymous type which saves the overhead of having to be named and explicitly referenced.

In general, anonymous types are identified by the lack of a *type* in an element (or attribute) declaration, and by the presence of an un-named (simple or complex) type definition as child.

1.4.5 Simple Types

New simple types are defined by deriving them from existing simple types (built-in's and derived). In particular, we can derive a new simple type by restricting an existing simple type. The *simpleType* element defines and name the new simple type. The *restriction* element indicates the existing (base) type, and identifies the "facets" that constrain the range of values.

XML Schema defines twelve facets. Among these, the *enumeration* facet is particularly useful and it can be used to constrain the values of almost every simple type, except the *boolean* type.

1.4.6 Occurrence Constraints

In some cases, it is required to enforce the cardinality constraints on the appearance of elements in schema. *Minoccurs* and *maxoccurs* attributes allow to specify the cardinality constraints. An element is required to appear when the value of *minOccurs* is 1 or more. The maximum number of times an element may appear is determined by the value of a *maxOccurs* attribute in its declaration. If a value for only the *minOccurs* attribute is specified, it is necessary to ensure that its value is less than or equal to the default value of *maxOccurs*, i.e. it is 0 or 1. Similarly, if a value for only the *maxOccurs* attribute is specified, it must be greater than or equal

to the default value of *minOccurs*, i.e. 1 or more. If both attributes are omitted, the element must appear exactly once.

1.4.7 Default values:

Default values of both attributes and elements are declared using the *default* attribute, although this attribute has a slightly different consequence in each case. Default attribute values apply when attributes are missing, and default element values apply when elements are empty. The *fixed* attribute is used in both attribute and element declarations to ensure that the attributes and elements are set to particular values. Note that the concepts of a *fixed* value and a *default* value are mutually exclusive, and so it is an error for a declaration to contain both *fixed* and *default* attributes.

1.4.8 Global Elements and Attributes

Global elements, and global attributes, are created by declarations that appear as the children of the *schema* element. Once declared, a global element or a global attribute can be referenced in one or more declarations using the *ref* attribute. The declaration of a global element also enables the element to appear at the top-level of an instance document.

There are a number of caveats concerning the use of global elements and attributes. One caveat is that global declarations cannot contain references; global declarations must identify simple and complex types directly. A second caveat is that cardinality constraints cannot be placed on global declarations i.e. global declarations cannot contain the attributes *minOccurs*, *maxOccurs*, or *use*.

1.4.9 Annotations

XML Schema provides three elements for annotating schemas for the benefit of both human readers and applications. The *documentation* element is the recommended location for human readable material and *xml:lang* attribute with any *documentation* element indicates the language of the information.

The appinfo element can be used to provide information for tools, stylesheets and other applications. Both *documentation* and *appinfo* appear as subelements of *annotation*, which may itself appear at the beginning of most schema constructions i.e *schema*, *element*, *attribute*, *complex type*, *simpletype* .

1.4.10 Attribute groups

Using an attribute group can improve the readability of schemas, and facilitates updating schemas because an attribute group can be defined and edited in one place and referenced in multiple definitions and declarations. An attribute group may also contain other attribute groups.

1.5 Outline of the thesis

First chapter gives the project motivation and provides background needed to understand the implementation details of our scheme. Chapter 2 briefly discusses some of the related work done in this area and also discusses different markup languages. Chapter 3 gives overview of our XSI tool and the phases of XSI tool. In chapter 4, we discuss implementation details of our proposed Deposit contract model. Finally, we conclude in the fifth chapter with a consideration of the future enhancements to the project.

In appendix-A, we have given the algorithms that used in XSI, Appendix-B concludes with the comprehensive list of constructs which has done and which has yet to complete.

Chapter 2

RELATED WORK

2.1 Markup Languages

2.1.1 RuleML:

The **Rule Markup Language** (**RuleML**) is a markup language developed to express both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks. It is defined by the **Rule Markup Initiative**, an open network of individuals and groups from both industry and academia that was formed to develop a canonical Web language for rules using XML markup and transformations from and to other rule standards or systems.

2.1.2 XBRL

eXtensible Business Reporting Language (XBRL)[9] is a financial specification which is initially focusing on company filings and reports. On an international level, the major complexity with company reports at present is that each country has its own accounting standard. In the United States of America (USA), the United States of America (US) generally accepted accounting principles (GAAP) is used, while in the United Kingdom (UK) it is the UK GAAP, Australia has an Australian GAAP, etc. Each accounting standard requires a different XBRL "taxonomy". What differs in each case are:

- The list of defined accounting items;
- The rules on how lower level items are added/subtracted/multiplied to give higher-level items.

XBRL Framework

The main ideas in the XBRL conceptual framework are items and taxonomies.

Items: In the XBRL framework, the most fundamental concept is that of the item. An item is meant to correspond to a fact often but not necessarily a numeric fact that is being reported with respect to a given period of time about a given business entity. For example, the fact that the company whose ticker symbol is SAMP reported revenues of $7m for the year 1998 is an item. This is an example of a numeric item. An example of a non-numeric item would be a paragraph

of text describing the principles of consolidation used to combine reports from the subsidiaries of SAMP. Although the latter is not numeric, this is nevertheless a fact being reported with respect to a given period of time (1998) about a given business entity (SAMP).

XBRL defines a syntax in which many different kinds of facts can be represented and their contexts defined in such a way that software applications can efficiently and reliably find, extract, and interpret relevant facts in their appropriate contexts.

Tuples: It is often the case that facts must be joined together to be understood. A tuple, like a row in a database table, is a grouping of facts. For instance, the name, age and compensation of a director of a company should be grouped together to be correctly understood.

Groups: In XBRL, a group is a set of related items that can appear in any order and can be interspersed among other text and elements in any XML document. The root of an XBRL Instance Document is a group, which may, in turn, contain other groups. There is, therefore, no "XBRL document type" as such. It is possible in principle to embed an XBRL item in any document, such as a press release that is otherwise formatted in HTML.

Elements and Taxonomies: An equally important part of the XBRL framework is the concept of an element and its relationships to other elements within taxonomy. In XBRL, the notion of a taxonomy element is represented by the notion of an element within an XML XML taxonomies are "vocabularies" or "dictionaries" created by a group in order to exchange information. In the future, there will be many XML taxonomies used to describe all types of data. What will drive the development of these taxonomies are industry groups, governmental agencies and other organizations.

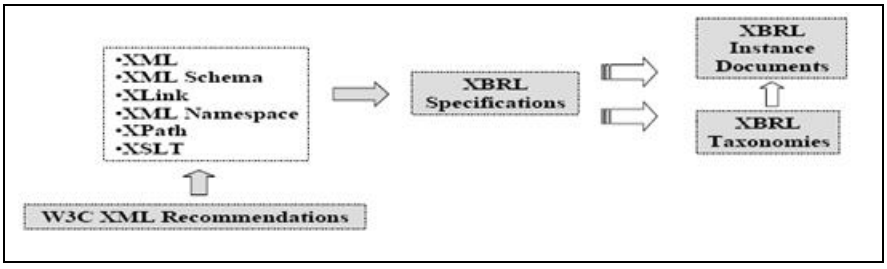

Figure 2.1 XBRL Taxonomy

XBRL taxonomy components (files)

There are six basic files that comprise XBRL taxonomy:

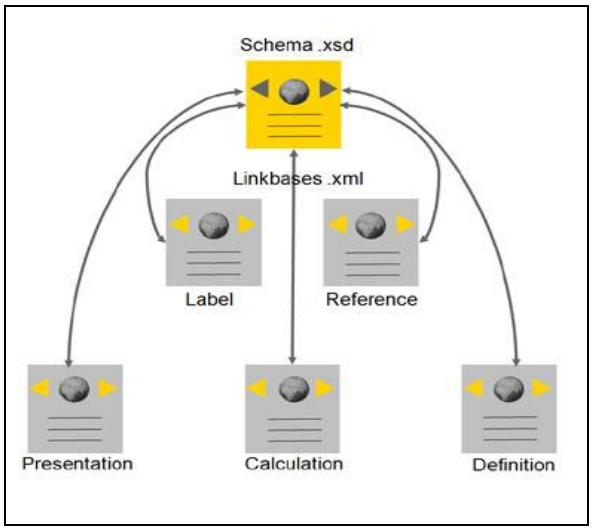

Figure 2.2 XBRL taxonomy is a combination of schema and linkbases

Schema

The XBRL taxonomy schema file defines the actual concepts that make up XBRL taxonomy. It gives their names, their data types, period type, whether you can report about them etc. The properties of concepts (elements) are defined in the XBRL specification.

Linkbases

An XBRL linkbase file contains the explicit relationship definitions between the concepts defined in the XBRL schema. There are five linkbase

- The **label** linkbase allows the user to attach labels with different roles and languages to a given concept
- The **reference** linkbase the user to attach external information (sources) to concepts
- The **presentation** linkbase defines how concepts are nested and ordered
- The **calculation** linkbase defines how values of concepts should sum up from one to another
- The **definition** linkbase allows the user to define additional semantics

2.1.3 MathML:

Mathematical Markup Language (MathML) [10] is an application of XML for describing mathematical notations and capturing both its structure and content. It aims at integrating mathematical formulae into World Wide Web pages and other documents. It is a recommendation of the W3C math working group.

2.1.4 IFX

Interactive financial exchange (IFX) is an XML messaging specification, which supports commercial and personal banking. IFX messages can be used to manage accounts and specify transfers of money between banks, businesses, and customers. IFX also handles billing and payments. This is an important part of commercial banking, but not investment banking, so IFX is a commercial rather than a financial XML format. IFX is based in part on OFX.

2.1.5 OFX

Open financial exchange (OFX) is an earlier commercial and personal banking specification that has migrated from standard generalized markup language (SGML) to XML (by removing support for implicit end tags). It has been extended to support tax forms, and there are also plans to extend it to support financial planning and insurance.

2.1.6 FpML

Financial Products Markup Language (FpML) is a set of financial specifications which is initially focusing on transactions of over-the-counter financial instruments. These are financial products which are not traded via an exchange, but directly between two financial institutions, such as banks.

2.1.7 MDDL

Market data definition language (MDDL) is a financial information specification produced by the financial information services division (FISD), part of the software and information industry association (SIIA). MDDL 1.0 was released at the start of November 2001, and supports the publication of snapshots and historical time-series of equity prices, financial indices, and mutual fund data.

2.1.8 swiftML

Society for worldwide inter-bank financial telecommunication (SWIFT) is an existing electronic messaging system used by major banks. The messages are being converted to XML under the name "swiftML". SWIFT are closely involved with international standards organization (ISO) 15022, on which the latest set of SWIFT messages is based, so swiftML is also closely aligned to ISO 15022. swiftML will be merged with FIXML as part of the ISO 15022 XML effort.

2.1.9 FIXML

Financial information exchange (FIX) is a non-XML financial transaction protocol, which aims to be vendor-neutral. The FIX consortium is composed of a group of banking and financial institutions who view themselves as clients rather than vendors. Financial information exchange markup language (FIXML) has been announced as the XML-isation of the existing FIX protocol (messages). The intention is that, for an interim period, both traditional FIX messages and FIXML will be supported in parallel until FIX is eventually deprecated in flavor of FIXML. FIXML will be merged with swiftML as part of the ISO 15022 XML efforts.

2.1.10 RIXML

Research information exchange markup language (RIXML) is an investment research specification which focuses on metadata rather than on the way that research reports are structured. The intention of RIXML is not to provide a way of writing investment research content using XML, but to provide a standard attachment that can be used with any media type to indicate the nature of the content. One of the key uses of XML in financial specifications is to allow metadata to be associated with content so that the best possible filtering and ranking of the available information can be done. RIXML is unusual in completely externalizing the metadata from the content, but effectively delivers its users' major value-add (filtering) without requiring any change to the content formats they use, so it does not interrupt existing workflows.

2.1.11 ISO 15022

A major non-XML specification has contributed to MDDL and other financial XML specification is ISO 15022. This provides a standard set of (>10k) data fields for financial information and (~100) messages for financial transactions. The data dictionary and catalogue of messages are maintained on ISO's behalf by SWIFT, a banking industry co-operative. An effort

has been started to define a direct XML version of ISO 15022, one which supports not only ISO 15022 but also the existing (non-XML) SWIFT and FIX transaction protocols.

2.1.12 eEbXML

ebXML is a general e-business XML specification, not specific to the any financial industry. it is likely to play a growing role in the financial world. Financial services are often divided into "view" (information) and "do" (transactions); the "do" half of financial XML is about financial transactions. Due to the sums of money and the risks involved, financial transactions typically have more demanding requirements for speed, validation, authentication, and security than do general business transactions. However, once the global ebXML infrastructure is in place, it would be very surprising if that infrastructure were not suitable for at least some financial transactions and the migration could well continue from there if the needs of the financial world drive the performance and functionality of ebXML implementations. So, expect to see ebXML take a not insignificant role in financial transactions in the future, though be aware that it is too early in ebXML's life to say where the first uses of ebXML for finance might occur.

2.2 Generation of User Interfaces from Composite Schema

The recent W3C XML technologies are providing a common document structure for exchanging data that makes it easily portable to any platform and device. A feasible solution can be achieved using these technologies. By this approach, user interface is generated dynamically from an XML schema document. And also, it is relatively easy to transform XML schema into GUI rendering Java swings that are also understood by most clients.

In this approach, GUI is only described implicitly through the XML schema. By doing so, the user experience of the application is tied directly to the application's data model. Changes in the data model are reflected in the user interface via a "push of a button", thus eliminating the need for tedious rewriting of presentation code. One manual step in the implementation process becomes superfluous.

2.2.1 JAXFront

JAXFront [11] is a commercial product that is implemented based on the above approach. It is a technology used to render electronic forms in multiple UI channels (Java swings, HTML

and PDF) on the basis of an XML schema that acts as a business model. It is built upon XML user interface (XUI) framework and XML schema. XUI framework allows schema authors to specify the template or a layout of GUI in a separate document. The format of this document is XML complained syntax. It describes how to display certain GUI components corresponding to schema elements, and to specify layouts for certain GUI components. The XUI document uses XPath expressions on XML schema elements to format and apply layouts on specific parts of XML schema.

The fundamental goal behind the development of this product is to reduce dependence of business personnel on application developers and also to minimize the tedious process involved in writing presentation code. It is still evolving in order to provide support for all of the elements present in schema vocabulary and to handle special cases such as importing other schema into existing schema.

2.2.2 Netbryx

Netbryx [12] is also another commercial product built upon the above solution. But, its scope is limited to only interface and for only Microsoft platform and PCs. It is developed in .NET platform and so, it cannot be portable to other platforms. It is not capable of producing GUI for all possible cases of schema documents. It is not handling of the some major elements of schema vocabulary such as choice, simple content, enumeration, complex content etc.

2.2.3 Microsoft InfoPath

Microsoft InfoPath (full name **Microsoft Office InfoPath**) is an application used to develop XML-based data entry forms, first released as part of the Microsoft Office 2003 suite of programs in late 2003 and later released as part of Microsoft Office 2007. Initially given the codename XDocs, the main feature of InfoPath is its ability to author and view XML documents with support for custom-defined XML schemata. It can connect to external systems using XML Web services through MSXML and the SOAP Toolkit, and back-end and middle-tier systems can be configured to communicate by using Web services standards such as SOAP, UDDI, and WSDL. Additionally, because InfoPath documents are raw XML, it is possible to directly repurpose the data in other XML processors.

InfoPath is based on Extensible Markup Language (XML). When we design a form template, InfoPath creates an '.xsn' file, which is a cabinet (.cab) file that contains the files

necessary for the form to function, such as XML Schema (XSD) and XSL Transformation (XSLT) files. When a user fills out a form in InfoPath, the data in that form is saved or submitted as industry-standard XML. However, user doesn't have to know anything about XML to design a form template or fill out a form. XML can make it easier for your organization to repurpose the data that it collects by using forms.

Example: A single InfoPath form template for trip reports can be used to provide XML data to a customer relationship management system, a petty cash system, and a travel planning system.

InfoPath provides several controls (e.g. Textbox, Radio Button, Checkbox, etc.) to present these data in data sources to end users. For data tables and secondary data sources, "Repeating Table" and other repeating controls are introduced. For each of these controls, actions (called "rules") can be bound in. A rule defines a specific action that will be performed under certain conditions.

Example: A simple rule could be: "Set field 'Total' to 100 when number in field 'field1' changes". All the data stored in InfoPath forms are stored in an XML format, which is referred to as the "data source". InfoPath can also use other data sources, such as Microsoft SQL Server or web services.

Chapter 3

XSI TOOL

3.1 XSI Overview

XSI (XML Schema based universal user Interface) is a tool under development in IDRBT. An XSI generates the user interface dynamically from composite XML schema documents across a wide variety of devices and platforms. XSI gets data from the user through the interface, creates XML document with elements from the schema and the user-entered data and validates the generated document against the XML schema. It generates GUI in java swings.

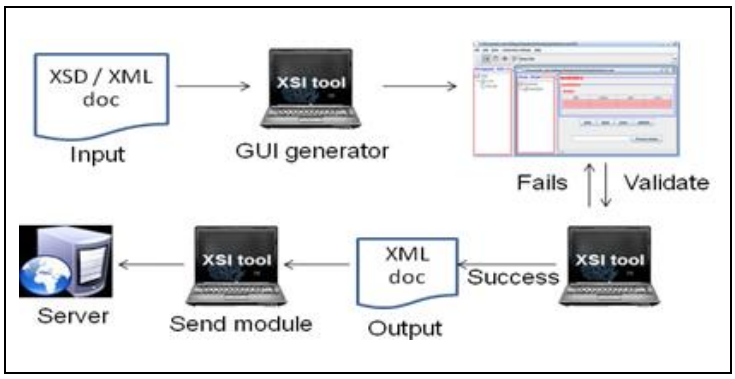

Fig 3.1 Overview of XSI tool

Phases in XSI

There are four phases in XSI, which is discussed in the below sections

3.2 Parsing XSD document

Most of existing parsers are providing support only for XML documents to parse and process. So, we thought that first the schema can be transformed into an instance, and then transform the instance into a user interface (UI). Although, an instance has the structure necessary to build a UI, it lacks crucial typing information. There would be no way to construct radio buttons, combo boxes and text boxes corresponding to the type of elements and to know constraints such as length of certain field, allowed patterns for a field or element if an instance is used to create the UI. In addition, there is no way to express *choice* elements in an instance document.

By noting all these problems, we proceeded directly to transform schema to UI due to the fact that schema documents do have same syntax as that of XML documents. So, schema is parsed with DOM parser bundled with Apache Xerces-J by assuming it as an XML document. The resultant of this step is a DOM tree representation of nodes which correspond to elements and attributes of given document. In DOM, both elements and attributes are treated as nodes. The nodes represent the schema vocabulary containing description about instance data nodes, but not the XML data.

The nodes of DOM tree are to be interpreted according to their context i.e. functionality, structure and etc. Then they are to be transformed into instance without values as well as GUI components using DOM API. We discuss underlying logic in transforming schema to GUI in the next section.

3.3 Transformation from XSD to GUI

Every XSD document does have one root element schema, also called as document element. The schema node contains global elements, global types, groupings and annotations as immediate children that include annotation, element, complexType, simpleType, group, attributeGroup nodes. One of global elements is the actual root element in the instance document. The following conditions need to be checked in selecting a root element for instance if there are many global elements. It must not be an empty element i.e. element containing only content but not any child elements and attributes.

If more than one global element is present (one with simple type and other with complex type), then the element of complex type is to be taken as root element for instance. It should not be referred from other elements using ref attribute.

Once the root element of instance is identified, it is inserted into a new DOM tree (or document) say Instance created using 'DocumentBuilderFactory' class. It will serve as DOM representation for XML instance for the given schema and also allows the application to insert data when data is entered later.

We designed a recursive algorithm to navigate the entire DOM tree of schema document until all the nodes are transformed into GUI components and whole XML instance is created. For 'convTo' method, the global element identified as root node is passed as input. The 'convTo' method has one handler for each and every element of schema vocabulary. The node passed as parameter is an element node, so it is handled by 'element_handler' method.

3.3.1 Element_handler:

In this handler, elements are classified into two types. They are global elements and non-global elements. The distinction is that global elements don't have ref attribute and occurrence constraints. So, they are handled separately. Initially, element_handler is called with global element i.e. root element. To process this element, the handler has to identify what type of element it is. If the type is predefined it is handled itself otherwise it direct to corresponding handler .

<element name="Name" type="xs:string" />	

Figure 3.2: Element having only content and type is 'String'

<element name="ChequeBookReqd" type="xs:boolean" />	

Figure 3.3: Element having only content and type is 'Boolean'

The handler first checks for type attribute of global element i.e. document element. If type attribute is not present, it will look for child (i.e. complexType) and then it calls complexType_handler to further explore sub elements. Otherwise, it finds the node with the name as specified in type attribute and that node is a complex type definition defined globally. For non-global elements, the handler calls Lookfor_type_attribute, Look-for_ref_attribute methods, which return the complex type definition node/referenced node if the current element is complex type or referencing another element. Only label component is added to interface, but it will not add text box for data entry. If the returned values are null, it indicates that element is simple type. So the label and text box components are added to the interface. The value of name attribute is inserted into the Instance tree in both cases.

3.3.2 ComplexType_handler: In this handler, the structure of complexType is analyzed. Generally, every complex type has one of the following three basic structures:

(a) Sequence: P-> abc i.e. consecutive elements, denoted by the sequence element.

(b) Repetition: B-> b* i.e. elements that occur n times (with 0 <= min <= n <= max <= infinity), specified by the minOccurs, maxOccurs attributes in element of the schema document respectively.

```
<xs:element name="MarksDetails" type="marks"
minOccurs="2" maxOccurs="4"/>
<xs:complexType name="marks">
   <xs:sequence>
      <xs:element name="Qualification"type="xs:string"/>
      <xs:element name="PerOfMarks" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>
```

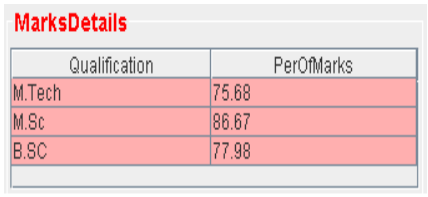

Figure 3.4 Element having 'min' and 'max' Occurrence

(c) Alternative: D->(e|f), denoted by choice elements **Arbitrary order:** The sub elements of a tag can appear in any order in the instance document. After processing these structures, it will look for more attributes and if there are any attributes or attribute groups are present, then it will invoke attribute_handler or attributegroup_handler for further transformation.

3.3.3 Sequence_handler:

It will iteratively loop through all sub elements to invoke corresponding handlers for each element type. It may also contain sequence, choice elements as sub elements within it. Generally, the sub elements of this node are element node, sequence node or choice node.

```
<xs:element name="StudentForm">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FirstName" type="xs:string"/>
      <xs:element name="LastName" type="xs:string"/>
      <xs:element name="ChequeBookReqd"
type="xs:boolean"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 3.5: 'complexType' Element and followed by 'sequence'

```
<xs:element name="internationalPrice">
   <xs:complexType>
      <xs:complexContent>
         <xs:element name="currency" type="xs:string"/>
         <xs:element name="value"    type="xs:decimal"/>
      </xs:complexContent>
  </xs:complexType>
</xs:element>
```

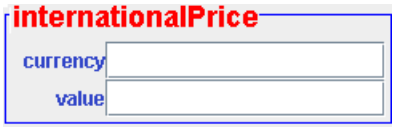

Figure 3.6: 'complexType' Element and followed by 'complexContent'

3.3.4 SimpleType_handler:

SimpleType element defines a simple type and specifies the constraints and information about the values of the attributes or text only elements. An element without child tags or attributes are termed as simpleType elements. By using simpleType, new simple types can be derived from existing simple types (built-in or derived). It is used to constrain the values of existing type. A simpleType element can occur as child of element or attribute and it can also be defined as a global type. In this way, it is handled and transformed into interface.

```
<xs:element name="Name" type="PersonName">
  <xs:simpleType name="PersonName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][.]([a-z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

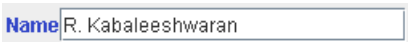

Figure 3.7: 'simpleType' element

3.3.5 Annotation_handler:

Annotation tag in schema document is used to provide some documentation that is useful for the processing of an application in order to process certain elements in schema and its instance. Annotation tags usually appear as first child of element, schema, and complexType elements as it is given in the following part of schema. If this element is having any child, it will invoke the Transform method with child node i.e. documentation as parameter.

3.3.6 Choice_handler:

Choice element allows only one of its child elements to appear in the instance document. In this handler, values of all sub elements' name attribute are added to combo box component of GUI. But, it won't add to instance DOM tree. Whenever the user selects one of the items in combo box list, then it adds to the DOM tree. And also, if the selected sub element is again a complex type element, their GUI components are added to interface by repainting the screen. Choice element may also contain another choice element as its child. In that case, logic becomes complex. It is still in evolution stages to derive efficient way for handling choice elements.

```
<xs:element name="Customer" type="cust"/>
<xs:complexType name="cust">
   <xs:sequence>
      <xs:group ref="custname"/>
      <xs:element name="Address" type="addType"/>
      <xs:element name="City" type="xs:string"/>
      <xs:element name="Country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="addType">
  <xs:choice>
    <xs:element name="Office" type="xs:string"/>
    <xs:element name="Res" type="xs:string"/>
  </xs:choice>
</xs:complexType>
<xs:group name="custname">
  <xs:sequence>
    <xs:element name="FirstName" type="xs:string"/>
    <xs:element name="LastName" type="xs:string"/>
  </xs:sequence>
</xs:group>
```

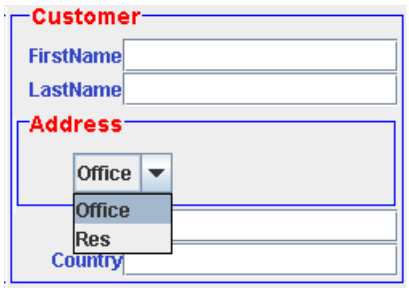

Figure 3.8: 'choice' element

3.3.7 AttributeGroup_handler:

AttributeGroup element is used to group only attributes but not elements. This handler is having the same functionality as that of group_handler. The group of attributes is added to instance DOM tree during transformation. For attributes also, a label component and a text box component is added to the interface.

```
<xs:element name="Item">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
    </xs:sequence>
  <xs:attributeGroup ref="ItemDelivery"/>
  </xs:complexType>
</xs:element>
<xs:attributeGroup id="ItemDelivery">
   <xs:attribute name="weightKg" type="xs:decimal"/>
   <xs:attribute name="productID" type="xs:string"/>
</xs:attributeGroup>
```

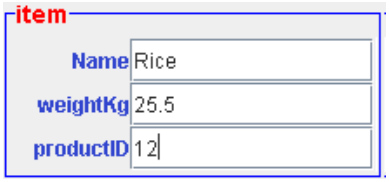

Figure 3.9:'attributeGroup' element

3.3.8 Group_handler:

```
<xs:element name="mail">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="Address"/>
      <xs:element name="Subject" type="xs:string"/>
     <xs:element name="Body" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:group id="Address">
  <xs:sequence>
    <xs:element name="From" type="xs:string"/>
    <xs:element name="To" type="xs:string"/>
 </xs:sequence>
</xs:group>
```

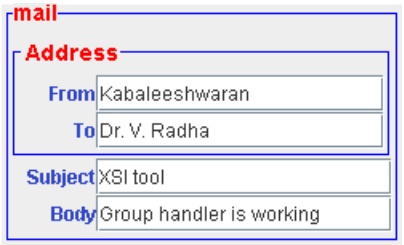

Figure 3.10: 'group' element

Group element allows a group of elements to occur either all of them or no element in the instance document. In this handler, it checks whether ref attribute is present or not. If ref attribute is defined, it will get the referenced node by calling Lookfor_ref_attribute method and transforms the child elements of that referenced Group element. If it is not, then it transforms its sub elements. The following schema snippet requires the above two handlers to handle this particular case.

3.3.9 Restriction_handler:

Restriction element in schema document is used to indicate the existing (base) type, and to identify the "facets" that constrains the range of values. There are twelve facets available in schema specification. These are: pattern, enumeration, maxlength, minlength, length, maxInclusive, minInclusive, maxExclusive, minExclusive, TotalDigits, FractionDigits and Whitespace.

Among these facets, enumeration is mostly used because it can be used to constrain the values of almost every simple type except boolean type. A restriction element can have more than one facet on the simple type. By considering all these options, we implemented one handler for each facet. These handlers based on their appearance in schema documents will be called from the restriction_handler itself.

```
<xs:element name="Gender">
   <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:enumeration value="Male"/>
        <xs:enumeration value="Female"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

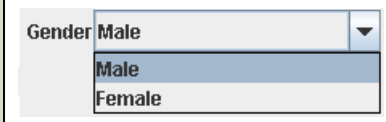

Figure 3.11: 'restriction' element

3.3.10 Processing XPath Expressions on XML through XSI

XPath is a language for finding information in an XML document, used to navigate through elements and attributes in an XML document. It is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. In popular use (though not in the official specification), an XPath expression is often referred to simply as an XPath. We used the JAXP API for processing the XPath expression on XML document.

3.3.11 XBRL Element handling in XSI

XBRL element has additional attributes to the W3C recommended element.

Example

```
<element name="Assets" id="Assets" periodType="instant" balance="debit"
abstract="false"   substitutionGroup="item" type="monetaryItemType"/>
```

Figure 3.12: attributes in XBRL doc

XBRL recommended using those attributes periodType and balance for every XBRL element. A value of the attributes, periodType is either instance or duration, balance is either debit or credit and substitutionGroup is item or tuple.

3.4 Generation of XML instance

In case of XSI as service approach, the data entered in various fields of interface will be passed as parameters to server. The passed parameters are given names containing prefixes in the order of root node to current node separated by colon. This type of naming convention helps us in tracking received parameters. We are collecting all received parameters into Enumeration

class. The parameters in this class are stored in random order rather than the order in which these parameters appeared in the interface.

Each received parameter is a pair of parameter name and parameter value. The parameter name is divided into tokens using '*stringTokenizer'* class. Each token represents a node name in the partially created instance DOM tree (i.e. without values). For each parameter, it will navigate from root of the DOM tree as per tokens in parameter name until the end of the tokens is reached. Then the parameter value will be inserted as text node in the DOM tree. This is repeated until all parameters are inserted into DOM tree of *Instance*.

In case of Java Swings version of XSI (XSI for Thick Clients), the value entered in the text box is instantly inserted using 'InsertNode' method, which is invoked from action listener (focus listener) of the text box.

3.5 Validation

We used JAXP API as part of Xerces parser to validate the instance document with the corresponding schema. We implemented a '*SchemaValidator'* class that takes XML instance and its schema document as parameters. Then it reads the XML instance using SAX API and validates it with schema. We kept this class as optional because we have not implemented handlers for certain elements schema vocabulary.

3.6 Serialization

Serialization is a process of creating a file in disk from DOM document in memory. That is, the contents of DOM tree are stored in the specified file. This process is also called as *marshalling*. We used '*XMLSerializer'* class to serialize XML instance document from the *Instance* DOM tree. By this class, we serialized as a 'DOMserializer'. We also set certain parameters for this class such as output format details, file name and etc.

Chapter 4

PROPOSED MODEL AND SOLUTION

4.1 Proposed Deposit Model in XSI

Model based on the XML technology for better transparency, accuracy and accountability for the parties involved in the deposit contract is presented. An overview of the deposit model is depicted in the Figure 4.1.

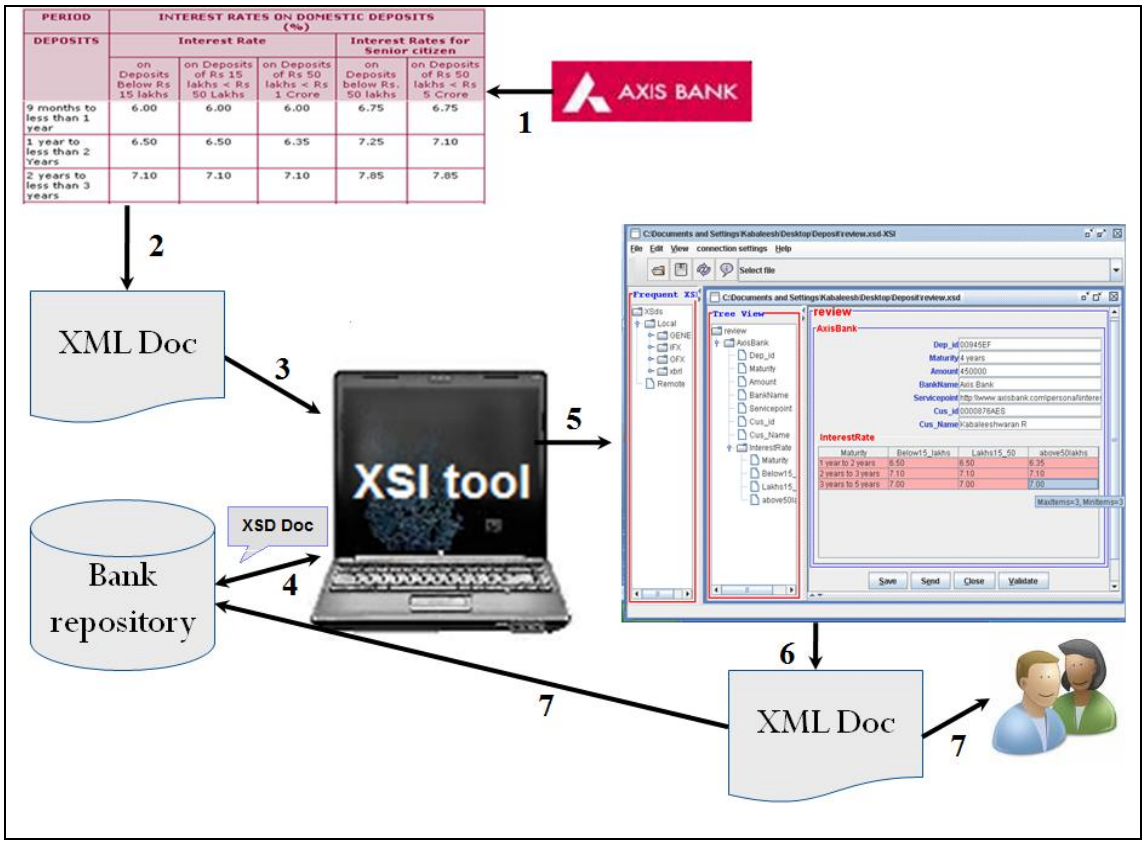

Figure 4.1: Overview of Deposit contract model

4.1.1 Banks Advertisement:

Timely bases bank advertises the Interest rate for Deposit financial product. Developer converts the advertisement into the xml document and a basic XML Schema document which is used to create GUI. While creating the XML document schema Location is included in it using

'schemaLocation'. Text box or combo box are not feasible enough to represent an element which occurs more than once, instead table is used to represent it.

```
<?xml version="1.0" encoding="UTF-8"?>
<oldDeposit schemaLocation="Bank XSD
repository/Fixed_deposit_rates_2007_Feb\Deposit.xsd">
   <include schemaLocation=" Bank XSD
repository/Fixed_deposit_rates_2007_Feb\Deposit.xsd"/>
   <Axisbank>
      <Deposit_id/>
      <MaturityTime/>
      <Amount/>
      < BankName/>
      <ServicePoint/>
      <Cus_id/>
      <Cus_Name/>
      <Issue Date/>
<Saving_InterestRate Maturity="9months_to_lessthan_1year"
AmountRange="Saving_InterestRate_below_15_lakhs">6.00</Saving_InterestRate>
      <Saving_InterestRate Maturity="9months_to_lessthan_1year"
AmountRange="Saving_InterestRate_15_to_50_lakhs">6.00</Saving_InterestRate>
      <Saving_InterestRate Maturity="9months_to_lessthan_1year"
AmountRange="Saving_InterestRate_above_50_lakhs">6.00</Saving_InterestRate>
      <Saving_InterestRate Maturity="1year_to_lessthan_2year"
AmountRange="Saving_InterestRate_below_15_lakhs">6.50</Saving_InterestRate>
      <Saving_InterestRate Maturity="1year_to_lessthan_2year"
AmountRange="Saving_InterestRate_15_to_50_lakhs">6.50</Saving_InterestRate>
      <Saving_InterestRate Maturity="1year_to_lessthan_2year"
AmountRange="Saving_InterestRate_above_50_lakhs">6.35</Saving_InterestRate>
      <Saving_InterestRate Maturity="2year_to_lessthan_3year"
AmountRange="Saving_InterestRate_below_15_lakhs">7.10</Saving_InterestRate>
      <Saving_InterestRate Maturity="2year_to_lessthan_3year"
AmountRange="Saving_InterestRate_15_to_50_lakhs">7.10</Saving_InterestRate>
      <Saving_InterestRate Maturity="2year_to_lessthan_3year"
AmountRange="Saving_InterestRate_above_50_lakhs">7.10</Saving_InterestRate>
   </Axisbank>
</oldDeposit>
```

Fig 4.2: Advertisement in the form of XML doc

4.1.2 Searching for basic XSD corresponding to the XML document

To the XSI tool the converted XML document is given as input to generate GUI. XSI searches for the schemaLocation in the corresponding XML document that refers to the service provider's repository. Bank repository maintains XSD for every service and looks after the updating in timely manner.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xs:element name="Axisbank">
      <xs:complexType>
         <xs:sequence>
```

```
            <xs:element name="Deposit_id" type="xs:string"/>
            <xs:element name="MaturityTime" type="xs:string"/>
            <xs:element name="Amount" type="xs:float"/>
            <xs:element name="BankName" type="xs:string"/>
            <xs:element name=" ServicePoint " type="xs:anyURI"/>
            <xs:element name="Cus_id" type="xs:string"/>
            <xs:element name="Cus_Name" type="xs:string"/>
            <xs:element name="Saving_InterestRate" type="IR" minOccurs="30"
maxOccurs="unbounded"/>
          </xs:sequence>
       </xs:complexType>
   </xs:element>
   <xs:complexType name="IR">
          <xs:attribute name="Maturity" type="xs:string" use="required"/>
          <xs:attribute name="AmountRange" type="xs:string" use="required"/>
   </xs:complexType>
</xs:schema>
```

Figure 4.3: basic XSD for Deposit contract Advertisement

4.1.3 Parsing XSD document and Generating GUI

The schema node contains global elements; one of the global elements is actual root element in the instance document. Once the root element of instance is identified, XSD is parsed with DOM parser bundled with Apache Xerces-J by assuming it as an XML document. As a result a DOM tree representation of nodes containing elements and attributes of given document. The nodes represent the schema vocabulary containing description about instance data nodes, but not the XML data. The nodes of DOM tree are to be interpreted according to their context i.e. functionality, structure and etc. It will serve as DOM representation for XML instance for the given schema and also allows the application to insert data when data is entered later. We designed a recursive algorithm in 'convertTo' method to navigate the entire DOM tree of schema document until all the nodes are transformed into GUI components.

```
<?xml version="1.0" encoding="UTF-8"?>
<FixedDeposit   schemaLocation="Bank XSD repository/Fixed_deposit_rates_2007_Feb">
    <Deposit_id> 00945EF </Deposit_id>
    <MaturityTime> 4 years </MaturityTime>
    <Amount> 450000 </Amount>
    <InterestRate>
       <fixed>  4.5  </fixed>
    </InterestRate>
    < BankName> Axis Bank < /BankName>
    <ServicePoint>   BanksURL   </ServicePoint>
    <Cus_id> 0000876AES </Cus_id>
    <Cus_Name>  R.Kabaleeshwaran  </Cus_Name>
    <Issue Date> March 10 2007  </Issue Date>
</Fixed Deposit>
```

Fig 4.4: Customer Details in XML doc

4.1.4 Validation and Creation of XML Instance

Customer submits the required information in the UI, further this information is inserted as an instance (XML document) of XSD document. Validation engine validates the XML document against its XSD document. This is a verification check performed to ensure that data provided by user conforms to the schema. If validation fails, the interface is refreshed and the user has to fill data with indicated changes.

4.1.5 Sending Instance document

The XML instance document of the deposit model will be send to the customer through mail or by hand. The customer can save the document in a trusted device for further reference and simultaneously it is send to the Bank Repository.

4.2 XSI – Developer side

XSI is a java-based tool developed with the help of the external java archives (jar) provided by Xerces [14]. Xerces has given java API for parsing of XML &XSD documents in the form jar files. XSI constitutes a total of 13 classes,

4.2.1 Combolistener – This class is designed as a listener for the JcomboBox (combo box is used if input is XSD document). Every element in the document will have a corresponding combo box in GUI. Its responsibility is to extract the value given by user and update DOM (DOM is a kind of data structure to hold the output i.e., XML document. Every input has a corresponding DOM with it.).

4.2.2 ComboRecentlist – XSI has a feature called Auto completion, which enables the user to select the values, which were entered earlier (in the past) for the same input document. It maintains buffer (in the form of file i.e., to be non-persistent) to store all the values for a particular element. For example, say element class / student / name, then every value entered for the element name will be stored in the file *name.txt* and this file is stored in directory of student, which was stored in directory class.

4.2.3 Converter – XSI is designed to accept those XSDs, which has a global element. If an XSD does not have a root element and instead has series of definitions then XSI would not accept the document. So converter has designed to convert non-compatible XSI inputs to compatible XSI input by grouping all the definitions in the document to one

global root element with its name its file name. Converter is an independent program, which is not part of XSI but serves a plug-in for XSI.

4.2.4 CreateTable – This class is written to handle the elements, which tend to occur more than once. If an element comes more than once then usual combo box is not sufficient to hold multiple items. So, we preferred a table to hold the elements. We have provided two keyboard shortcuts for the table like

- Cntrl+insert to add a new row
- Cntrl+delete to delete an existing row

These works can also be done by the right click of mouse, which pops up two items saying *Add row* and *Delete row*. Whenever a row is added or deleted corresponding changes have to be made in DOM of input.

4.2.5 CustomFilter – Users are allowed to give two kinds of input to the XSI. One is XSD and other is XML. Users are provided with a file chooser (JfileChooser in java, used as input dialog box to choose a file) where he selects the file. CustomFilter is used to provide the user all the kinds of file he need. For example, if he wants to open an XSD file then only XSD files are visible.

4.2.6 Editor –This class is starting point for the XSI. It takes the responsibility of initial design. Things like menu bar, tool bar, split pane (to separate two entities/panels in a flexible way, 'JSplitPane' in java) are designed in Editor. It takes care of forwarding input document to XSI (class which parses and generates GUI) and pasting the internal frame returned on its slot.

4.2.7 IncludesQueue – As schemas become larger, it is often desirable to divide their content among several schema documents for purposes such as ease of maintenance, access control, and readability. This can be achieved by using include construct. This class maintains a queue to hold all the include constructs *schemaLocation* values.

4.2.8 SendXML – The ultimate goal of XSI is to enable the user to exchange the newly created XML instances to the opposite party. This class does that by using web services framework. This class takes the web service description and finds the match for it. Then it sends the document.

4.2.9 SortVector – This class is used to sort the vector of strings. It makes uses of quicksort algorithm as it is best rated algorithm for sorting (since its time complexity is O (nlogn for n items).

4.2.10 StringSort – This class uses 'SortVector' to sort the strings. Once sorting is done, it will store the sorted vector into a file.

4.2.11 Validation – One of the features XSI provides is validating of input given by the user after GUI has presented. Users may mistakenly insert incorrect entry to a combo box, text box or table cell. This class takes all the information given by the user and validates it, if any errors found then this class would put them in validation part of the internal frame via a text area.

4.2.12 Xmldata – When the input is given in the form of an XML document, then XSI must load the corresponding values of fields before presenting GUI. 'Xmldata' serves this purpose. It is helpful in getting values for a given elements. After fetching the values XSI will load them into the corresponding element's text box.

4.2.13 XSI - This class holds the responsibility of parsing and understanding of given XSD document. It has all handlers needed to generate GUI for the document. It will also generate a tree for the XML document. It will paste the widgets on the internal frame that was passed by the class Editor. XSI also keeps the values entered in the widget to provide auto-complete option; it saves the item's values in text files to achieve this feature. It uses 'convTo' (a method that is used to redirect to corresponding handler like 'element_handler', 'complexType_hand' and so on) to redirect particular node to its corresponding handler, and handlers are written for every possible node type. It takes the help of Validation class to validate the values entered by the user

These are all the classes that are used in the making of XSI. Editor acts as a starting point for the tool's execution (It contains main method). Then the Editor will hand over the input documents to class XSI where main functionality is achieved. XSI takes element by element from the input document and parses them. In order to extract the attribute values it uses the method 'getAttrValue(node, attributeName)' and to get the node reference for the attribute value it uses 'getAttrNode(node, attrbuteName)'. At the end of the parsing it will group them and presents on GUI. In the process of generating GUI class XSI uses all the classes listed above.

4.3 Contributions

- Text box or combo box are not feasible enough to represent an element which occurs more than once, instead table is used to represent it.
- Created XML document to represent Table with data.
- Implemented to retrieve the data from the table using ‘fillTableData’ and ‘tableFunction’ methods in ‘XSI’ class and ‘matchchild’ method from ‘xmldata’ class.
- If we include the XSD file in the XML document using ‘schemaLocation’, XSI is capable of search the corresponding schema Location. If it finds the location XSI generates GUI corresponding to XSD. If it doesn’t, XSI display the error message to indicate the error.
- Since attribute can be added to the instance by specifying default values, but element cannot. (They must appear to receive a default value). So after representing the table format (ex., Interest rate for deposit) using attribute, customer enters data by selecting the particular Interest rate corresponds to the ‘amount range’ and ‘duration’. While creating an Instance document XSI takes only selected Interest rate (element), not the attributes (like ‘amount range’ and ‘duration’) from the XML.

Chapter 5

CONCLUSION AND FUTURE SCOPE

5.1 Experimental Setup

We implemented on Windows NT system running on 2.4GHz, Pentium IV, 256 MB machine. We used Java Swings of j2sdk1.4.2 version [15]. We also used Xerces-j v2.6.2 API, developed by Apache as DOM parser, for validating XML documents with the associated schema, for serialization and also to extract and add Namespaces from/to schemas and its instances. This version also extends JAXP (Java API for XML Processing), developed by Sun Micro systems, which provides programmable interface for validating XML documents with XSD.

5.2 Screen shots

5.2.1 Deposit contract model

If we open the XML document in XSI, (from Fig 4.2) it will display all the Interest Rate (given by the bank) in the table format. Customer submits the required information in the UI provided by XSI. Customer has to select the Interest rate from the table corresponding to values of Amount and Maturity from his information, and save it as Instance document. The following Fig 5.1 represents the above scenario, in which the amount is 450000 and Maturity is 4 years. Customer selects interest rate 7.00 corresponding to the Amount and maturity.

5.3 Conclusion

The XSI software is the part of the system is the smart GUI solution not only supports the XSD documents as input but also XML documents. It validates the data without human interventions. It supports several Markup languages like XBRL, IFX, and OFX. The main aim of the system is to make open for adapting to XML Standards and enlighten the same methodology to take to other financial products like deposit, loans and so on.

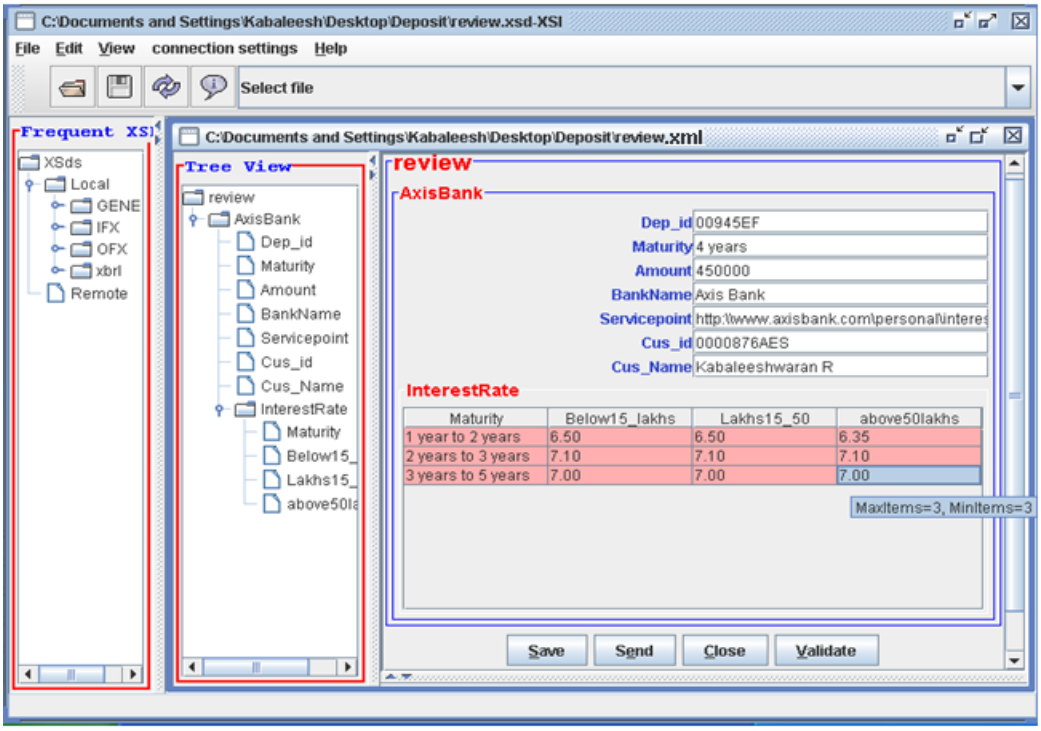

Fig 5.1 Deposit contract model

5.4 Future Scope of the project

XSI can be extended to support the following additional features:

1. The calculation part is right now embedded within XSI, which however has to be expressed in the form of XSD itself
2. Now tool is capable providing user interface for few markup languages (IFX, OFX and partially XBRL), trying to handle major markup languages like MathML, RuleML and so on.
3. Adjustment in the interface when the elements are less i.e., element count is less than 10
4. If schema document imports or includes elements from other schema documents using *any* element, then GUI components are also to be generated for these elements. Thereby, this tool can generate GUI for multiple schema documents.
5. In some cases, the XPath expression specified as part of unique element of schema element is to be processed. And additional information provided in *appinfo* element is to be handled by processing application.

6. It is required to ensure that namespaces from schema are correctly interpreted and then added to instance document, especially if a schema document derives from multiple schema documents.

In addition, the generation of arbitrary complex GUIs requires the specification of various declarative models thus it cannot be covered in whole by automated processes based on just schema data. It is required to specify complex declarative models and also to do certain types of validations such as cross field validation, processing of certain XPath expressions [17] and etc. XML Style sheet Language (XSL) specification [16] serves the purpose. In turn, the developers or business personnel have to define schema document and style sheet document for each service or form.

5.4.1 Native XML Databases

The structured data in web environment is being circulated in the form of XML documents. So, large collections of XML data need to be efficiently stored managed and queried using native XML databases. Our tool is to be integrated with native XML database products such as X-Hive DB [18] and Berkeley DB XML [19] in order to store and process the received XML instances through different modules of XSI.

5.4.2 XML Security Mechanisms

Security is the prime concern for the XML data circulated over the web. The traditional security mechanisms that apply to normal documents differ from the security mechanisms that apply to XML documents. But, the security features such as confidentiality, authentication, integrity and non-repudation remain same in both cases.

XSI can also be extended to incorporate security mechanisms by using XML Encryption APIs [21], XML signatures APIs [20,23] and XACML [22](Access control policies) in order to provide confidentiality and integrity of XML data circulated in our both approaches and authenticity of XML documents.

APPENDIX A

Algorithm 1: Vector rows filltabledata (Vector rows, String[] colHeader, int index)

Input: *colHeader* – contains all the elements name that used to represent as a column header

Index -the position of the algorithm

Output: *rows* - Vector which contains value of the row

```
Begin
    int rowct=0;
    while(rowct<tableCount)
    Begin
        String[] array1 = stringtokenize(colHeader[rowct]);
        xmldata x = null;
        x = new xmldata(array1, xmlfilename, true, noofrows); /* to retrieve the XML data */
        String[][] value = x.tablevalues;
       while(jj<value.length)
       Begin
           if(value[index1][jj]!=null)
               flagfiltab = true;
               singleRow =value[index1][jj];
               rows.addElement(singleRow);
               jj=jj+1;
               break; /* to terminate the loop */
        End
        rowct = rowct+1; /* increment the row count */
    End
    return rows;
End
```

Algorithm 2

String[][] **matchchild**(Node child, String array[], String tableName, int index)

Input: child – root node of the XML document

Array[] – contains all the first child in each level of DOM tree.

Index –the position of the algorithm.

Output: String[][] – returns the table values retrieved from the XML document

```
Begin
/* The node should not be null */
    While(child != null)
    Begin
  /* match the values from the root */
     if (child.getNodeName().equals(array[index]))
  /* search for the table name which contains max Occurs>1 also it should not be root,
  Check it whether node has attribute or not*/
      if(child.getNodeName().equals(tableName)
            AND child.hasAttributes() AND index>0)
    /* collect those attributes */
       NamedNodeMap nnm=child.getAttributes();
       while(i<nnm.getLength())
       Begin
          Node Att=nnm.item(i);
          /* Check the name of the attributes with names in the XSD */
            if(Att.getNodeName().equals(array[index+1]) OR
                  Att.getNodeName().equals(array[index+2]))
              Node val=Att.getFirstChild();
              value="";
              if(val!=null)
                 /* collect the values of the attributesin 2 dim array */
                 if(val.getNodeType()==Node.TEXT_NODE)
                    Atttablevalues[row][col]=val.getNodeValue().trim();
                    col=col+1;
          i=i+1; /* increment to the next attribute*/
```

```
            End
           /* collect the mixed content value from the DOM tree */
            Atttablevalues[row][col] = child.getTextContent();
            col=0;
            row=row+1;
          else
            matchchild(child.getFirstChild(),array,tableName,index+1);
    child = child.getNextSibling();  /* increment the child from each level*/
End
```

Appendix B

The above are all the keywords described by **W3C**. A bold item represents constructs, which have to be handled in the future. An italic item represents those, which were handled earlier during my senior time. And the remaining constructs are developed in this year.

XML Schema Elements	**XML Schema Attributes**
all	abstract (element declaration)
annotation	abstract (ComplexType)
any	**attributeFormDefault**
anyAttribute	**base (simple)**
appinfo	**base(complex)**
attribute	**block**
attributeGroup	**blockDefault**
choice	**default (element)**
complexContent	**default (Attribute)**
complexType	**elementFormDefault**
documentation	**final**
element	**finalDefault**
enumeration	**fixed(attribute, element, simpletype)**
extension	**form(attribute, element)**
field	**itemType**
group	**memberTypes**
import	maxOccurs
include	minOccurs
key	**mixed**
keyref	name
length	**namespace (element wildcard, imprt specification)**
list	**xsi:noNameSpaceSchemaLocation**
maxInclusive	**xsi:nil**
maxLength	**nillable**
minInclusive	**processContents(element wildcards, attribute wildcards)**
minLength	**ref**
pattern	schemaLocation (include, **redefine, import**)
redefine	xsi:schemaLocation
restriction	**subsitutionGroup**
schema	**targetNameSpace**
selector	type (element, attribute)
sequence	**xsi:type**
simpleContent	**use**
simpleType	**xpath**
union	
unique	

References:

[1] V.Radha, N.Pradeep Kumar, S.RamaKrishna, "**Generic XML Schema Definition to GUI Translator**", *International Conference on Distributed Computing and Internet Technology*-2005, LNCS 290-296.

[2] V Radha, S Ramakrishna " **A XML schems based user interface**" *Enabling Technologies for Smart Appliances*, ETSA 2004, Hyderabad, India

[3] Priscilla Walmsley, **Definitive XML Schema** (Second Edition), Prentice Hall PTR, 2002.

[4] T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler, "**Extensible Markup Language (XML) Version 1.0 (Second Edition)",** W3C Recommendation, *http://www.w3.org/TR/REC-xml/,* November 2008

[5] David C. Fallside, Priscilla Walmsley, "**XML Schema Definition (XSD) Version 1.0**", *http://www.w3.org/TR/xmlschema-0/*: W3C Recommendation October 2004

[6] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson and L. Wood. **Document Object Model (DOM) Level 1 Specification**. W3C Recommendation. October 1998. *http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/* .

[7] SAX - **Simple API for XML**, *http://sax.sourceforge.net.*

[8] David C. Fallside, Priscilla Walmsley, "**XML Schema Definition (XSD) Version 1.0**", *http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/,* W3C Recommendation, May 2001.

[9] Geoffrey Shuetrim David Vun Kannaon Hugh Wallis Phillip Engel, Walter Hamscher. **"Extensible business reporting language (xbrl) 2.1"** http://www.xbrl.org/2003/XBRL-WD-2003-04-23.pdf

[10] David Carlisle, Patrick Ion, Robert Miner and Nico Poppelier. **Mathematical Markup Language (MathML)** Version 2.0 (Second Edition) *http://www.w3.org/TR/MathML/* 21 October 2003

[11] **JaxFront** – *http://www.jaxfront.com*

[12] **NetBryx** – *http://www.netbryx.com*

[13] **MS Office InfoPath** - *http://office.microsoft.com/en-us/infopath/default.aspx*

[14] **Xerces2-j parser** - *http://xml.apache.org/xerces2-j/.*

[15] **Java API –** *http://java.sun.com*

[16] J. Clark, "**XML Style sheet Language Transformations (XSLT) V 1.0**" *http://www.w3.org/TR/1999/REC-xslt-19991116*, W3C Recommendation, November 1999.

[17] A. Bergulund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Simon. **XML Path Language (XPath) 2.0 Technical Report**, W3C Working Draft

[18] X-Hive DB Native XML data base – *http://www.x-hive.com/products/db/index.html.*

[19] *S.* Mullan and A. Nadalin. JSR 105: XML Digital Signature APIs. http://www.jcp.org/en/jsr/detail?id=105.

[20] A. Nadalin. JSR 106: XML Encryption APIs – *http://www.jcp.org/en/jsr/detail?id=106.*

[21] OASIS Consortium. eXtensible Access Control Markup Language (XACML) Committee Specification, Version 1.1 Available at *http://www.oasis-open.org/committees/xacml/.*

[22] M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon, "XML Signature Syntax and Processing", http://www.w3.org/TR/xmldsig-core/, W3C Recommendation, February 2002.