Towards Metadata Driven User Interfaces

Thesis
Presented in Partial Fulfillment of the Requirements for
The Degree of Master of Science in the
Graduate School of the Ohio State University

By
Rahul Mukhedkar, B.Tech
Computer Science and Engineering

The Ohio State University
2010

Thesis Committee:
Dr. Rajiv Ramnath, Advisor
Dr. Jay Ramanathan, Co-Advisor
Dr. Raghu Machiraju, Co-Advisor
Abstract

Every computer application needs a user interface (UI) to interact with the system. However, developing a user interface from scratch is time consuming, error prone and lacks flexibility and extensibility. UI frameworks make UI development easier but they only address look and feel aspects, not the behavior part of the UI because they are domain independent. We present an approach to address this problem effectively by shrinking the problem domain and reusing domain specific metadata to generate the UI for applications in a domain. This thesis examines this approach for creating customized user interfaces for clinical applications dealing with reports based on Health Level 7 standards.
Dedicated to my family
ACKNOWLEDGEMENTS

I would like to thank my family for their unwavering support not only in my graduate studies but also in all of my endeavors.

I would like to thank my advisors Professor Rajiv Ramnath and Professor Jay Ramanathan for their support and friendship throughout the development of this thesis and my graduate studies as a whole.

Lastly, I would also like to thank Strategic Thinking Industries, Professor Igor I. Malkiman and CETI at The Ohio State University for their part in influencing this work.
VITA

2007............................B.Tech, Computer Science & Engineering, Guru
Gobind Singh Indraprastha University, India
Jan 2008-Aug 2008........Software Tester, Hewitt Associates, India
June 2009-Sep 2009.......Software Developer, Strategic Thinking Industries,
Columbus, Ohio
Jan 2010- Present.........Software Developer, Office of Student Life, The Ohio
State University

Field of Study

Major Field: Computer Science & Engineering
TABLE OF CONTENTS

Abstract .................................................................................................................................................. ii

ACKNOWLEDGEMENTS....................................................................................................................... iv

VITA .......................................................................................................................................................... v

TABLE OF CONTENTS ............................................................................................................................. vi

List of figures .............................................................................................................................................. xi

Chapter 1 Thesis Organization and Problem Statement ................................................................. 1
  1.1 Organization of This Thesis .............................................................................................................. 1
  1.2 Problem Statement .......................................................................................................................... 2
    1.2.1 Need for Application Frameworks.......................................................................................... 2
    1.2.2 Shortcomings of Present Application Frameworks ................................................................. 3
  1.3 Our Approach................................................................................................................................... 4

Chapter 2 Related Research ................................................................................................................... 5
  2.1 Related Research ............................................................................................................................. 5
    2.1.1 The Role of Stylesheets ......................................................................................................... 5
    2.1.2 Present Commercial User Interface Frameworks Research.................................................. 6
2.1.3 Domain Specific User Interface Research ............................................. 6
2.2 Discussion ................................................................................................. 8

Chapter 3 Solution Design ............................................................................. 10

3.1 Introduction to Solution Design ................................................................. 10
  3.1.1 Purpose .................................................................................................. 10
  3.1.2 Background of Client - Strategic Thinking Industries (STI) ................. 10
  3.1.3 Electronic Health Record (EHR) and Health Level 7 standards ........... 11

3.2 Shortcomings of the Present Solution ......................................................... 12

3.3 Shortcomings of the ESB approach ............................................................ 13

3.3 Features of the New Design ....................................................................... 14

3.4 Motivation behind New Interface Design .................................................. 15
  3.4.1 Component Flow Diagram .................................................................... 15
  3.4.2 Discussion ............................................................................................. 17

3.5 System Architecture .................................................................................. 18
  3.5.1 Presentation and Business Logic Layer ................................................ 19
  3.5.2 Google Web Toolkit .............................................................................. 19

3.5.3 Data Layer ............................................................................................ 21

3.5.4 Enterprise Service Bus (ESB) ................................................................. 21
2. Installation .............................................................................................................. 41

2.1. Obtaining the Software ......................................................................................... 41

2.2. Supported Operating Systems ................................................................................ 41

3. Using the Dynamic Outbound Interface ................................................................. 41

3.1. Starting the Internet Explorer ............................................................................... 41

3.2. Login Page .............................................................................................................. 42

3.3. Map File page ........................................................................................................ 43

Appendix B: STI Programmer Guide ............................................................................. 47

Table of Contents .......................................................................................................... 48

1. Introduction ............................................................................................................... 49

1.1 Overview ................................................................................................................ 49

1.2 Prerequisites .......................................................................................................... 49

2. Installation ................................................................................................................. 50

2.1 Obtaining Google Web Toolkit ............................................................................. 50

2.2 Obtaining Chainbuilder ESB ................................................................................ 50

3. Client side code ....................................................................................................... 51

3.1 Introduction ............................................................................................................. 51

3.2 Classes used .......................................................................................................... 51
4. Server Side Code........................................................................................................52

4.1 Introduction...........................................................................................................52

4.2. Classes Used.................................................................................................53
List of figures

Figure 1 STI old framework [Strategic Thinking Industries] ........................................ 13
Figure 2 ESB user interface ...................................................................................... 14
Figure 3 New UI Design [Koppal, 2009] .................................................................. 15
Figure 4 Component flow diagram in ESB [Koppal, 2009] ................................. 16
Figure 5 System Architecture .................................................................................. 19
Figure 6 Overview of GWT API [Hanson and Tacy, 2007] .................................... 20
Figure 7 Example of ESB workflow [Chappell, 2004] ............................................ 22
Figure 8 Chainbuilder ESB product schematic [Chainbuilder] ......................... 24
Figure 9 Example of HL7 metadata file ................................................................. 25
Figure 10 elementDef metadata file ....................................................................... 26
Figure 11 User Interface ......................................................................................... 28
Figure 12 Proposed Metadata for Semantic Checking ........................................... 29
Figure 13 Proposed Metadata for User Profiles ..................................................... 30
Figure 14 Part of the Component flow diagram in XML ....................................... 32
Figure 15 Login page .............................................................................................. 42
Figure 16 Map file page ......................................................................................... 43
Figure 17 Map file page left side ............................................................................ 43
Figure 18 Map file page tab panel ........................................................................... 44
Figure 19 Map file page with all components enabled ........................................... 44
Figure 20 Illustration of mapping ........................................................................ 46
Chapter 1

Thesis Organization and Problem Statement

1.1 Organization of This Thesis

Chapter 1 discusses the importance of user interfaces and the advantages and shortcomings of present application frameworks used for developing user interfaces for applications.

Chapter 2 surveys existing research in the area of user interface development for various applications and specific domains.

Chapter 3 provides a detailed description of the prototype developed including a description of the utilized technology and the overall system architecture.

Chapter 4 presents a new design that extends the solution presented in Chapter 3 with enhanced use of metadata for user interface generation.

Chapter 5 presents conclusions and future work.
1.2 Problem Statement

1.2.1 Need for Application Frameworks

A vital part of every software application is its user interface (UI). A good user interface serves as a mediator between the users and the software to assists the users in retrieving the relevant output to their input data or actions. Poorly designed user interfaces will not be able to effectively perform the above function. In applications with graphical user interfaces, nearly 50 percent of source code lines and development time can be attributed to the UI [Myers and Rosson, 1992]. Developing a UI is complex due to the complexity of the task and the complexity of the technology. Thus, building a UI is costly and time consuming, even as developers are under constant pressure to develop applications as quickly as possible. Since the UI is a large part of every software application, automating it (or some portion of it) will reduce the development and debugging time of the overall software, and reduces the cost and improves the quality of the software.

To cater to the problem of rapid software development, numerous application frameworks are widely used in the industry today. A framework is a reusable, "semi-complete" application that can be specialized to produce custom applications [Johnson and Foote, 1988; Fayad and Schmidt, 1997]. Software
design paradigms have shifted from procedural to object oriented approaches and so have frameworks. Object oriented application frameworks provide benefits like modularity, reusability, extensibility and inversion of control to the developers [Fayad and Schmidth, 1997].

Application frameworks can be categorized on the basis of their scope: system infrastructure frameworks, middleware integration frameworks and enterprise application frameworks. We are interested in enterprise application frameworks as they address broad application domains where the applications are expensive to develop.

1.2.2 Shortcomings of Present Application Frameworks

Application frameworks are used to develop applications for a wide range of domains that include the healthcare sector, banking sector, etc. They provide a generic UI framework for all the classes of problems they address, leaving the developer responsible for creating any custom code that is needed. Our goal is to automate the custom application development process as much as possible by utilizing the domain knowledge encoded within a framework.
1.3 Our Approach

Domain knowledge is the knowledge about the environment in which the target system is to operate. Domain knowledge can be:

i) Developer and end-user knowledge about the domain.

ii) The application framework knowledge about the domain embedded in the metadata contained in the application framework.

We seek to take advantage of these two sources of domain knowledge to efficiently build the application specific UI.

The domain knowledge in the metadata is of two kinds: workflow or behavioral knowledge of the application, and structural knowledge, which is the relationship between the entities in the domain. This domain knowledge can be represented in a machine readable form (such as XML), that can be used to generate the user interface.
Chapter 2

Related Research

2.1 Related Research

2.1.1 The Role of Stylesheets

The area of user interface (UI) frameworks has been the subject of extensive research. Stylesheets such as CSS and XSL are widely used to define the style and structure of web pages. The advantages of using stylesheets are speed, maintainability, customization, consistency and portability [Henry, 2003]. The problem with stylesheets is that the developer cannot specify any dynamic functionality; e.g. such as to obtain system date. Java Server Pages (JSP) [Bergsten, 2003] allows the developer to add computation to HTML pages, thus requiring only a HTML editor to monitor these changes. The problem with both Stylesheets and JSP is that they do not take any advantage of any existing behavior or structural knowledge.
2.1.2 Present Commercial User Interface Frameworks Research

Products from Oracle, Sybase, and Microsoft are based around a palette using which a developer can specify the layout and orientation of basic GUI components [Lakshman, 2000]. This allows the developer to drag and drop the required components and adjust their orientation. The developer then has to manually write code to define the interaction between the components and the functionality behind the UI.

These commercial products only support the layout and display properties of visual components [Mamrak and Pole, 2002] and also do not consider any behavior and structural knowledge about the domain.

2.1.3 Domain Specific User Interface Research

One noteworthy work that makes use of behavior and structural knowledge to generate the UI is the Acuity project [Mamrak and Pole, 2002]. In this work web based forms are automatically generated for updating a web accessible database. The Acuity project uses XML as its specification language, and takes advantage of behavior and structural knowledge to automate code for event handlers and listeners - thus reducing the task of the developer. This is contrary to typical application frameworks where the developer has to write a part of the
code manually for event handlers and listeners. Mamrak and Sinha [1999] also provide a case study using the Acuity project showing reduction in both development and debugging time for new applications using object oriented framework technologies.

Similar work has been done by Shimomura [2004] for web applications in which an image-oriented page transition framework is used to model web applications as a set of transitions of web pages. This work uses visual components that make it easy to write processes for web page navigation. The advantage of this framework is that the developer does not need to write any code to receive or analyze the common submitted data. The data needed for the page transfers is stored in appropriate variables automatically through automated code generation. Again, this saves a lot of development and debugging time.

Similar work has been done for multimedia database applications by Shirotay and Iizawa [1997], by asking the user to provide the metadata of the source database into their system.

Browne [1997] introduced a model-based approach by decomposing UI design into separate models, each of which may be declaratively specified. Once specified, automated tools integrate the models and generate the necessary

Finally, note that the problem of UI development is not limited to traditional desktop applications but also to handheld devices. Nilsson [2009] discusses issues with user interfaces for mobile applications and problems that need to be addressed due to the limitations of screen size and computational capabilities in these devices.

2.2 Discussion
Our literature survey showed that there is considerable amount of work being done in this field. In all of the above examples it will be seen that the problem has essentially been handled effectively by narrowing the application domain.

In our approach, we studied the different application frameworks that aim to solve this problem. We then tried to reuse the metadata from the application frameworks. Using this metadata, which contained structural and behavioral domain knowledge we generated a UI for an application in the medical domain. We also found that while this UI was useful, the metadata used was not
sufficient. We then identified additional sources of metadata and how they may be used to generate interfaces that provide additional and more refined functionality.
Chapter 3

Solution Design

This chapter describes the user interface (UI) we developed for the application in the medical domain.

3.1 Introduction to Solution Design

3.1.1 Purpose
An intranet web based application was developed for Strategic Thinking Industries (STI). The purpose of this application is to generate an XML map file that served as an input to an enterprise service bus (ESB) in order to generate electronic health records (EHR) in Health Level 7 (HL7) format from records in a database. The user interface of the application was intended to provide a customized user-friendly interface for the generation of the map file.

3.1.2 Background of Client - Strategic Thinking Industries (STI)
STI specialized in providing industry leading information technologies to all facets of hospital and healthcare facilities [Strategic Thinking Industries]. STI already
3.1.3 Electronic Health Record (EHR) and Health Level 7 standards

The Health Management System Society [The Healthcare Information and Management Systems Society] defines EHR as “a longitudinal electronic record of patient health information generated by one or more encounters in any care delivery setting.”

HL7 is defined as “a not-for-profit, ANSI-accredited standards developing organization dedicated to providing a comprehensive framework and related standards for the exchange, integration, sharing, and retrieval of electronic health information that supports clinical practice and the management, delivery and evaluation of health services”[Health Level Seven International]. The 7 in HL7 is a reference to the seventh (application) layer of the Open System Interconnection (OSI) Reference Model. The HL7 vision is to be the best and most widely use standard in healthcare.
3.2 Shortcomings of the Present Solution

We analyzed the present STI solution and came up with these shortcomings:

a) The present solution is tightly coupled to the specific integration. As a result, STI has to build a new interface for every new clinical system they must support. Furthermore, because it is a monolithic application any enhancement in software components requires the entire application to be upgraded, requiring considerable amount of time and effort [Koppal, 2009].

b) The present interface generates static reports i.e. users do not have control to configure and customize reports. Therefore, as requirements of the reports change appropriate code changes have to be made in the application, thus once again further development costs and leading to upgrades [Koppal, 2009].

c) Extensive IT training is required to operate the present interface correctly.
3.3 Shortcomings of the ESB approach

The previous solution from STI needed custom programming to address the integration needs. Using an Enterprise Service Bus (ESB) addressed the integration customization problem, and provided other functionalities. However, the ESB complex user interface (figure 2) could not be used by the expected user base of nurses and other medical staff. Essentially, the extra functionality in the ESB made the UI more complex. Thus, a custom UI was needed that
provided the benefits of the ESB and was more focused towards the usage of the expected user base.

Figure 2 ESB user interface

3.3 Features of the New Design

Our new design had the following features:

1) A three-tier architecture consisting of a presentation layer, the business logic layer, and the data layer. This architecture provided a loosely coupled solution.
2) Reports were made configurable. Drag and drop functionality was implemented to assist users with this configuration.

3) A user friendly web application was developed to minimize IT training.

4) Security was implemented through simple login mechanism.

Figure 3 New UI Design [Koppal, 2009]

3.4 Motivation behind New Interface Design

3.4.1 Component Flow Diagram

The previous STI solution suffered the need to custom code to address every integration need and the resulting data transformations desired. An ESB solved this integration problem as it provided an easy and efficient means to integrate the various data sources and produce the needed file transformations. However, the ESB brought a new problem that we did not anticipate. The ESB
configuration comes with an interface that is generic and non customizable which makes it efficiently not usable. Furthermore, the ESB provided other functionalities that may not be required by the user which ended up confusing the user. Therefore, we needed a better solution. To investigate how a better solution might be developed we looked for additional metadata within the ESB. We then came across to component flow diagram that represents:

Figure 4 Component flow diagram in ESB [Koppal, 2009]
The above diagram is significant as it describes the entire assembly of components required for report generation. The different components and their respective functions are described below:

1. Script: Responsible for retrieving data from the database. This component is configured to fetch data at specified regular intervals.
2. Sequencer: Controls the flow of data and sequence of operations. Incoming edges to the sequencer indicates the component that provides input. Edges with numbers indicate the sequence in which operations are performed.
3. JDBC (Java Database Connectivity): Responsible for connecting to the database and retrieving data from it.
4. Transformation: Responsible for file transformation.
5. File Writer: Responsible for writing the result of file transformation.

Using a component flow diagram ensures that the functionalities is not hard coded and loosely coupled.

3.4.2 Discussion

The component flow diagram made us realize the role of ESB for report generation. ESB provides the mapping between the source (database) and the
destination (HL7 fields). Also, this mapping is one to one. Two types of metadata to be considered are:

1. Metadata that gives information about the database field mapped to the HL7 field.
2. Process metadata that gives information about the execution process, starting from the retrieval of records from database to the writing of the converted HL7 result.

The important observation for (1) is that the metadata is in form of XML file that goes from one component to another. What changes frequently in the component flow diagram is the map file, not the different components. Thus, the need to configure the different components over and over again is not required.

Thus, if we provide a user interface with all the data fields available the, only thing left to be generated would be the map file. Therefore, our UI consists of two main components, one panel for the fields of the database and the other panel for the HL7 fields. Users should be able to view both the datasource and HL7 fields simultaneously and map the database field to the desired HL7 field.

3.5 System Architecture

This section details the system architecture and the technologies used:
3.5.1 Presentation and Business Logic Layer

The Presentation layer consists of a web browser. The Business Logic Layer was developed using Google Web Toolkit (GWT).

3.5.2 Google Web Toolkit

GWT is a development toolkit for building and optimizing complex browser-based applications [Google Web Toolkit]. GWT consists of libraries in Java which when compiled converts the code from Java to JavaScript during runtime. Thus, GWT allows one to develop JavaScript applications in Java. This is accomplished by
the Java-to-JavaScript Compiler, the most important part of GWT. This compiler translates Java code into JavaScript that runs in the user’s browser. This frees the developer from the burden of rewriting JavaScript to suit the peculiarities and lack of standards support in all the various browsers in use [Smeets, Boness and Bankras, 2008]. Currently, GWT supports Internet Explorer 6 and above, Firefox 1.0 and above, Safari 2.0 and above, and Opera 9.0 and above. GWT is used for the business logic layer because it is an open source framework having a powerful plug-in for Eclipse [Google Web Toolkit]. GWT also has other advantages, such as providing a platform for faster Ajax applications, preventing memory leaks etc. [Hanson and Tacy, 2007].

Figure 6 Overview of GWT API [Hanson and Tacy, 2007]
3.5.3 Data Layer

The Data Layer consists of Microsoft Sql Server 2005 Standard Edition. GWT and ESB both use Java Database Connectivity (JDBC) drivers to connect to the database.

3.5.4 Enterprise Service Bus (ESB)

In an ESB, applications and event driven services are tied together in a Service Oriented Architecture (SOA) in a loosely coupled fashion. This allows them to operate independently from one another while still providing value to a broader business function [Chappell, 2004]. Applications (and integration components) in ESB are decoupled from each other, and connected together through the bus as logical endpoints that are exposed as event-driven services. Thus, an ESB can efficiently provide central configuration, deployment, and management of services that are distributed across the extended enterprise.
3.5.5 Chainbuilder Enterprise Service Bus (CBESB)

We are using Chainbuilder ESB, a Java Business Integration (JBI) compliant enterprise service bus developed by Bostech Corporation. We use the community version of CBESB that is free. CBESB provide services like message parsing, data transformation, content based routing, sequencing and JDBC access to relational databases [Chainbuilder]. CBESB was chosen because of its simple, user friendly interface and drag and drop functionality for designing a flow which allows developers to visually create and ensemble an SOA. Key features of the CBESB were:

1) Open standards based: CBESB supports technology standards such as WSDL, HTTP, SOAP, JDBC, HL7 and EDI.

Figure 7 Example of ESB workflow [Chappell, 2004]
2) Integration with disparate sources: CBESB supports database integration and message integration.

3) Easy to develop.

4) Easy to extend: CBESB contains an extension framework called “User Point of Control (UPoc) that allows extending pre built components by plugging in Java code components.

5) Monitoring and Control: CBESB provided an Administrative Console which is an Ajax based web interface which was used to perform remote monitoring, administration and configuration of alerts at run time of JBI components and service assemblies.
Figure 8 Chainbuilder ESB product schematic [Chainbuilder]

3.6 Description of User Interface

See Appendix A and B.

3.7 HL7 Metadata file

A typical HL7 XML file is shown below:
The different components of HL7 metadata file are

1) Element name: The content of this tag is used as the label to display the HL7 field on the user interface.

2) ElementDef: Each field in HL7 has a specific value in it, for example: data field can only have database field, numbers can only have numbers etc.

Figure 9 Example of HL7 metadata file
Another XML file containing elementDef is looked up to find its definition.

For example xml file for "ST" is defined as:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<mdl:definition xmlns:mdl="http://cbesb.bostechcorp.com/ml/1.0">
  <mdl:description>String</mdl:description>
</mdl:definition>
```

Figure 10 elementDef metadata file

3) maxLength: This defines the length of the input for a specific HL7 field.
Chapter 4

Enhanced Use of Metadata for User Interface Generation

As it is obvious from Figure 11, the user interface (UI) provided in chapter 3 was still complex for the end users. Firstly, checking the correctness of mappings generated by users was limited to checks of field length; however, users often mapped fields incorrectly because they did not understand the semantics of each field. Secondly, we observed that users only mapped a subset of the attributes, with the subset depending on the role that the user played in the organization. Therefore displaying all the attributes to every user was not required and only added clutter to the user interface. Thirdly, the attributes were displayed in the order they were present in the map file, and not in order of relevance.
We next show how structural domain knowledge may be enhanced to address these issues. Note that the enhancements are necessary because the existing metadata does not encode the knowledge necessary for the above. We also show how metadata encoded behavioral knowledge may be used to generate the overall behavior of the user interface.
4.1 Structural metadata

4.1.1 Semantic checks

The above diagram is an example of structural metadata that may be used for semantic checking. The diagram conveys that a patient can be identified by name and SSN but age is merely an attribute of a patient from which he or she cannot be identified. This metadata can be used to provide semantic checks that will help restrain user from erroneously mapping a (data source field to an HL7 field) that will result in an error.

Figure 12 Proposed Metadata for Semantic Checking
4.1.2 User profiles

The above diagram shows how metadata representing structural domain knowledge may be used to restrict the mappable fields presented to a user. In this diagram, a staff member is classified as either medical or non-medical, and further, as a doctor or a nurse. A medical staff nurse is allowed only to read preconfigured HL7 reports while a medical staff doctor may read and configure new HL7 reports. Further, a medical staff member is interested in data source and HL7 fields related to patient healthcare whereas a non medical staff insurance agent is interested in fields related to billing and payment. By representing these roles and their interests in metadata, the system provides a
knowledge driven means of restricting visibility and accessibility to only relevant fields in a role-driven manner.

4.1.3 Removing order dependency

Note that in the current implementation, the HL7 fields were displayed in the same order as in the HL7 schema files. Ideally, the system should display the more relevant HL7 fields earlier in the list. Thus a dentist may prefer to see the fields related to teeth first while an ENT specialist will prefer to see fields related to ear, nose and throat first. This ordering (and grouping) may also be driven by meta-data similar to that in the figure above.

4.2 Behavioral metadata

4.2.1 Dynamic creation of user interface

In section 3.4.1 we discussed the role of the component flow diagram in specifying the transformation from source data to target data. The component flow diagram thus contains behavioral domain knowledge, in that it encodes the overall workflow of the application. In fact, because the component flow diagram has an encoding in XML, the different components needed in the UI may be identified by programmatically inspecting the component flow diagram.
For example, in the case of the component flow diagram (CFD) used for this application, the first component encountered is the Java Database Connectivity (JDBC) component. Thus, the user interface must begin with a screen whose function is to configure the datasource that will contain the position for datasource and the respective JDBC driver. Next, the CFD represents an overall mapping from the source data to target data; thus the user interface may be auto-generated to consist of a panel that shows elements of the source data source, a panel showing elements of the target data source and a visual paradigm to map one to the other.

Figure 14 Part of the Component flow diagram in XML
Chapter 5

Conclusion and Future Work

5.1 Conclusion

The User Interface (UI) is vital for every software application, whether it is a web based or a stand-alone application, and developing it takes a significant amount of time. We have attempted to show that a solution to this problem is the use of metadata to generate the UI for limited domains. Shrinking the problem domain and using domain knowledge in the metadata are the two significant principles through which the generation of the UI may be automated.

5.2 Future work

We would like to extend the prototype and add the functionalities mentioned in Chapter 4.

Presently, this application does not support generating a formatted file such as .doc or .pdf. The reason for this is the ESB used in this application; Chainbuilder ESB (CBESB) does not provide any support for these formats. However, XML generation is supported by CBESB which can be used to produce .pdf reports.
There exist frameworks today that convert XML to PDF such as JasperReports [Jaspersoft]. Future work is to also study these frameworks and try to use their metadata to integrate their functionality into our existing solution.
References


Chainbuilder. *Chainbuilder CONNECT Community Edition*  
[http://www.chainforge.net/](http://www.chainforge.net/)


Google Web Toolkit. Google Web Toolkit Overview  
[http://code.google.com/webtoolkit/overview.html](http://code.google.com/webtoolkit/overview.html)


Health Level Seven International. *About HL7.* [http://www.hl7.org/about/index.cfm](http://www.hl7.org/about/index.cfm)
http://www.macromedia.com/macromedia/accessibility/pub/acc_sites_chap01.pdf


Smeets, Bram; Boness, Uri; Bankras Roald *Beginning Google Web Toolkit: From Novice to Professional*. Apress, September 2008


Strategic Thinking Industries. *Our mission*. 


Strategic Thinking Industries. http://www.st-industries.com/


Appendix A: STI User Guide
## Table of Contents

1. Introduction .................................................................................................................. 40
   1.1. Overview ................................................................................................................. 40
   1.2. Prerequisites ........................................................................................................... 40

2. Installation ...................................................................................................................... 41
   2.1. Obtaining the Software ......................................................................................... 41
   2.2. Supported Operating Systems ............................................................................... 41

3. Starting the Internet Explorer ......................................................................................... 41
   3.1. Starting the Internet Explorer ................................................................................ 41
   3.2. Login Page ............................................................................................................. 42
   3.4. Map File page ........................................................................................................ 43
1. Introduction

1.1. Overview

This guide is designed to help you quickly get the STI Dynamic Outbound Interface get working and run some sample implementations. Before starting with STI Dynamic Outbound Interface, it may be helpful to present a high level overview of the product.

- STI Dynamic Outbound Interface is a user interface designed to make generation and scheduling of Health Level 7 (HL7) reports user friendly, faster, easier and more reliable.
- STI Dynamic Outbound Interface is built on Google Web Toolkit platform and run over Apache Tomcat Server.

1.2. Prerequisites

The user should have a basic understanding of Internet Explorer and Windows XP. However, this user interface is designed to be self contained and easy to work with. Also, the sample implementations provide a good basis for learning how this user interface works.
2. Installation

2.1. Obtaining the Software

The Dynamic User Interface is the property of Strategic Thinking Industries. Please contact STI at sales@st-industries.com for a copy of this software. Also, one can contact C.E.T.I at The Ohio State University (http://ceti.cse.ohio-state.edu/) to obtain this software.

2.2. Supported Operating Systems

Presently the user interface is supported on Windows XP/Vista/7. Only Internet Explorer browser is supported.

3. Using the Dynamic Outbound Interface

This section provides an example and screen walkthrough for the Dynamic Outbound Interface.

3.1. Starting the Internet Explorer

Launch Internet Explorer from the Start Menu.

At the Internet Explorer screen, the following screen comes up. Please enter the following URL (http://localhost:9080/STI) and press enter.
3.2. Login Page

The following login page would appear

![Login page](image)

**Figure 15 Login page**

Please enter the username as “cpnuser” and password as “cpn”. (Username and Password both case sensitive) and click login.
3.3. Map File page

This is how the Map File page of the Dynamic User Interface looks. The main features you see on this page are:

- On the left side, a drop down list box with names of the tables from the database. In order to see more values in this column, please contact the administrator.
On the right side, a tab panel consisting of different message headers of the HL7 standard is shown. Please see the HL7 standard for further information.

![Figure 18 Map file page tab panel](image)

- After clicking on the different message header, this screen appears.

![Figure 19 Map file page with all components enabled](image)

If you have opened an existing map file, the contents of the message header would be displayed, as shown above.
If you have created a new map file from scratch, all the message header fields would be blank.

- This image illustrates how a field from the database can be dropped onto a message header. Here are a few things one must remember while performing this operation:
  - Once a table is selected, the choice cannot be redone. You would have to close the browser and start again with the correct table you want.
  - Only one table can be selected at one time.
  - Only one database value can be mapped in one HL7 message field.
  - The database value has to be dragged and dropped in order to be mapped; if written by the keyboard the value would be taken as a literal only.
Figure 20 Illustration of mapping

In the above picture, I have selected the PatientName First database field and dropped it on MSH08.

With this, you can input from the keyboard or drag and drop and make your own map file with your requirement.
Appendix B: STI Programmer Guide
Table of Contents

1. Introduction ................................................................................................. 49
   1.1. Overview ............................................................................................... 49
       1.1.1 Prerequisites .................................................................................. 49

2. Installation ...................................................................................................... 50
   2.1 Obtaining Google Web Toolkit ............................................................... 50
   2.2 Obtaining Chainbuilder ESB ................................................................. 50

3. Client Side code ............................................................................................ 51
   3.1 Introduction ............................................................................................ 51
   3.2 Classes used ......................................................................................... 51

4. Server Side code ........................................................................................... 52
   4.1 Introduction ........................................................................................... 52
   4.2 Classes used ......................................................................................... 53
1. Introduction

1.1 Overview

This guide is designed to help you with the programming environment setup for STI Dynamic User Interface and get you familiar with the code so one can start programming on the user interface. Before starting with the specific details, it may be helpful to present a high level overview of the programming environment.

- STI Dynamic Outbound Interface was programmed in Java using the Google Web Toolkit plug-in in Eclipse.

1.2 Prerequisites

The following software’s should already be installed on your machine

- Windows XP 32 bit operating system
- Java Development Kit 1.6 or higher
- Microsoft SqlServer 2005 Standard Edition
- Apache Tomcat Server
- Eclipse IDE
2. Installation

2.1 Obtaining Google Web Toolkit

Google Web Toolkit (GWT) is an open source set of tools that allows web developers to create and maintain complex JavaScript front end applications in Java. It is licensed under the Apache License version 2.0.

GWT plug-in for Eclipse can be obtained from http://code.google.com/eclipse/

Though one does not need Eclipse to work with GWT, it is encouraged to use Eclipse as the IDE.

2.2 Obtaining Chainbuilder ESB

Chainbuilder ESB is a Java Business Integration (JBI) compliant product which consists of a set of Eclipse GUI plug-ins, runtime server components and a Web-based Admin console. Its primary uses are in Service Oriented Architecture (SOA) environments and Enterprise Application Integration (EAI). We would be using the freely available version of Chainbuilder ESB.

Chainbuilder ESB can be downloaded from http://www.chainforge.net/download/index.html
3. Client side code

3.1 Introduction

This section contains details about the client side implementation for the Dynamic Outbound Interface.

3.2 Classes used

Here is the list of classes used for the client side implementation.

1) Test10.java
   - This is the main entry class for the application.

2) SimpleDropController1.java
   - This class extends SimpleDropController class which is used for drag and drop functionality.

3) GreetingServiceAsync.java
   - This class is used for Ajax calls to the server with GreetingService.java.

4) GreetingService.java
   - This class is used for Ajax calls to the server with GreetingServiceAsync.java.

5) FileDisplay.java
   - This class is used to display the existing map files.
6) Field.java
   - This class creates a composite widget consisting of a label and textbox which is used for each HL7 message header field.

7) CustomTextBox.java
   - This class extends a textbox class and is used to clear the textbox text upon double clicking.

4. Server Side Code

4.1 Introduction

This section contains information about the server side implementation for the Dynamic Outbound Interface.

Server side consists of the following components:

1) MSSQL (database)
2) Chainbuilder enterpriser service bus
3) Tomcat Apache Server
4) Server code
5) JDBC driver (Java Database Connectivity)
6) HL7 metadata files
7) Other images/resources used in the user interface
4.2. Classes Used

Server code is written using java. Here is the list of classes used for the server side implementation.

1) GreetServiceImpl.java
   - This is the main class on the server.

2) Filter.java
   - This class filters only the specific file extension from the rest of the files available on the server.

3) ProcessHandler.java
   - This class is used to refresh the java stack when Deploy and Run commands are executed.