Enterprise Information Architecture for the dissemination of Geothermal Data

Anshuman Mehta
Barcelona School of Informatics
Universitat Politècnica de Catalunya

A thesis submitted for the degree of
Master in Information Technology

2011 September
Abstract

Information systems in an organization are usually developed in silos and remain usable only within a certain part of the organization. One of the challenges of Service Oriented Architecture is to develop information systems with reusable components and services which are then composed to enable new services.

This thesis enables an information system for the dissemination of Geothermal Data by using Enterprise Integration Patterns in an Event Driven Service Oriented Architecture. The system developed with this architecture is loosely coupled and is highly scalable and flexible. This thesis gives details about the architecture, design and implementation of the system and the underlying technologies and concepts used to develop the system.
To my wife for her consistent help and guidance, my parents and sister for their continuous support and encouragement.
Acknowledgements

I am grateful to Fabian Moerchen for providing me an opportunity to research in Siemens Corporate Research and the freedom provided to work and develop on the thesis. I am indebted to Phillip Hussels for his constant guidance and technical discussions which strengthened the thesis. I would also like to thank the software development team in the GTDA project project at SCR for their valuable contributions to the project.

I would like to thank Toni Urpi for his guidance, Gemma Sese for her continuous support at UPC and Xavier Franch and other professors at UPC. I would also like to thank the Unitech International Organization for the opportunity to undertake this internship under their banner and for the delightful academic experience at ETH.

My special thanks go to my wife Indu for her understanding and love during the past few years. Her support and encouragement was in the end what made this dissertation possible. My parents, Om Prakash and Sunita, receive my deepest gratitude and love for their dedication and the many years of support during my studies. I would also like to thank my sister, Shaifali, for her knocks of guidance, support and love over all these years.
Contents

Glossary vii

1 Introduction 1
  1.1 Motivation ............................................. 1
  1.2 The Thesis ............................................ 2
  1.3 Contributions ........................................ 2
  1.4 Layout of the Document ............................... 3

2 Background 5
  2.1 Service Oriented Architecture ...................... 5
    2.1.1 Enterprise Service Bus .............................. 5
    2.1.2 REST ............................................. 6
  2.2 Enterprise Integration Patterns .................... 6
    2.2.1 Message ........................................... 6
    2.2.2 Message Channel .................................. 6
    2.2.3 Message Endpoints ................................. 8
    2.2.4 Message Translator ................................. 8
    2.2.5 Filter ............................................. 8
    2.2.6 Content Enricher .................................. 9
    2.2.7 Router ............................................. 9
    2.2.8 Splitter ........................................... 10
    2.2.9 Aggregator ........................................ 10
    2.2.10 Service Activator ................................. 10
    2.2.11 Channel Adapter ................................. 11
CONTENTS

3 Requirements  
  3.1 Project Requirements ............................................. 13  
  3.2 Overview ............................................................. 13  
  3.3 Use Cases ............................................................. 15  
  3.4 Automatization ....................................................... 18  

4 Architecture and Technology  
  4.1 Architecture .......................................................... 19  
  4.2 Services ............................................................... 20  
  4.3 Technology Survey .................................................... 21  
    4.3.1 Data Integration .................................................. 21  
    4.3.2 User Interface .................................................... 22  
    4.3.3 Search ............................................................ 24  
    4.3.4 File Repository .................................................. 24  
    4.3.5 Geospatial WebServices ......................................... 25  
    4.3.6 Security .......................................................... 25  
  4.4 Decisions .............................................................. 26  
    4.4.1 Spring Integration ............................................... 26  
    4.4.2 ExtJS ............................................................. 26  
    4.4.3 Solr ............................................................... 27  
    4.4.4 Custom VirtualFileSystem Implementation ..................... 27  
    4.4.5 Geoportal and GeoServer ....................................... 27  
    4.4.6 Shiro ............................................................. 27  
    4.4.7 Spring Batch and Spring Batch Admin .......................... 27  
    4.4.8 PostgreSQL and PostGIS ........................................ 28  

5 Design  
  5.1 Design ............................................................... 29  
    5.1.1 Application ....................................................... 29  
    5.1.2 Publication Pipeline ............................................. 30  
    5.1.3 ETL ............................................................... 31  
  5.2 Modules .............................................................. 33  
    5.2.1 Persistence ....................................................... 34  
    5.2.2 File System ...................................................... 35
CONTENTS

5.2.3 File Detection ............................................. 35
5.2.4 Integration .................................................. 37
5.2.5 Server ......................................................... 37
5.2.6 ETL Services ................................................ 38
5.2.7 ETL Batch Admin ............................................. 38
5.2.8 Security ......................................................... 38
5.2.9 Workflow ....................................................... 39
5.2.10 Geoportal ..................................................... 39
5.2.11 Solr .......................................................... 39

5.3 Packaging of Modules ........................................ 40

5.4 Database ........................................................ 40
5.4.1 FileSystem. .................................................... 40
5.4.2 Integration. .................................................... 41
5.4.3 Core. .......................................................... 42
5.4.4 Staging. ......................................................... 44
5.4.5 GtdaBatch. ..................................................... 44
5.4.6 Geoportal. ..................................................... 45
5.4.7 Security. ....................................................... 46
5.4.8 workflow ....................................................... 46

5.5 Decisions ......................................................... 47
5.5.1 UUID. .......................................................... 47
5.5.2 Integration Wrapper Services. .............................. 47
5.5.3 Channels. ....................................................... 47
5.5.4 Transaction Boundary. ....................................... 48
5.5.5 Persistence Units. .......................................... 48

6 Implementation .................................................... 49
6.1 The Chain Pattern .............................................. 49
6.2 Integration of Modules ......................................... 50
6.2.1 Upload a File .................................................. 50
6.2.2 Fetch Uploaded files ........................................ 50
6.2.3 Split the List of Messages and Filter Detected and Undetected Files ........................................ 52
CONTENTS

6.2.4 Transform the message ........................................ 54
6.2.5 Route the Messages to the appropriate Services .......... 54
6.2.6 ServiceActivators ............................................. 56
   6.2.6.1 File Detection ........................................... 56
   6.2.6.2 Validate Publications .................................... 58
   6.2.6.3 Load Bibliography ....................................... 59
   6.2.6.4 SOLR Publisher ........................................ 61
   6.2.6.5 GEOPORTAL Publisher ................................... 62
6.3 Security .......................................................... 63
6.4 ETL Batch Admin ................................................. 67

7 Evaluation .......................................................... 69
   7.1 Environment .................................................... 69
   7.2 Degree of Loose Coupling .................................... 70
   7.3 Throughput ..................................................... 73
   7.4 Degree of Reusability ........................................ 75
   7.5 Development Effort ........................................... 76
   7.6 Discussion ..................................................... 77

8 Conclusions ........................................................ 79
   8.1 Conclusions .................................................... 79
   8.2 Future Work ..................................................... 80

9 Appendix ............................................................ 81
   9.1 Plan .................................................................... 81
      9.1.1 Initial plan ................................................... 81
      9.1.2 Development plan .......................................... 81
      9.1.3 Variance ..................................................... 84
   9.2 User Screens ..................................................... 84

List of Figures ........................................................ 93

List of Tables .......................................................... 95

References .............................................................. 97
# Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CSW</td>
<td>Catalog Service for the Web</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Object</td>
</tr>
<tr>
<td>DOE</td>
<td>Department of Energy</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>EDA</td>
<td>Event Driven Architecture</td>
</tr>
<tr>
<td>EIP</td>
<td>Enterprise Integration Patterns</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GTDA</td>
<td>Geothermal Data Aggregation</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java Platform Enterprise Edition</td>
</tr>
<tr>
<td>JCR</td>
<td>Java Content Repository</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Messaging Service</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistence API</td>
</tr>
<tr>
<td>JS</td>
<td>Java Script</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>MAVEN</td>
<td>Software Project Management and Comprehension Tool</td>
</tr>
<tr>
<td>NGDS</td>
<td>National Geothermal Data System</td>
</tr>
<tr>
<td>RAD</td>
<td>Rapid Application Development</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RIA</td>
<td>Rich Internet Applications</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SDS</td>
<td>Software Design Specification</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SpEL</td>
<td>Spring Expression Language</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirement Specification</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign On</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URI</td>
<td>Universal Resource Identifier</td>
</tr>
<tr>
<td>WFS</td>
<td>Web Feature Service</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1

Introduction

1.1 Motivation

The Department of Energy (DOE) of the United States of America is funding projects in the area of assessment and exploration of geothermal energy sources. One such effort is the creation of the National Geothermal Data System (NGDS) that will integrate many data sources relevant for geothermal resource assessment and development. The goal for NGDS is to lower the exploration risk by making a wide range of relevant data available to stakeholders.

The Geothermal Data Aggregation (GTDA) project aims to significantly contribute to NGDS by submitting relevant data from a team of geothermal specialists. The GTDA project will include the creation of a centralized database to gather the data provided by the consortium. The database will be accompanied by a software system that supports the import and review of data by the consortium partners, automated processing for quality assurance and data analytics, and compliant web-services for submission of data to the NGDS. Successful completion of this project would result in multiple positive outcomes. Primary benefits will include support for developers of geothermal power plants by decreasing the costs of the resource identification phase and the risks of the exploration phase of developers. The project will rescue important data from deterioration or complete loss and provide a set of tools to be used by other parties to submit data to the NGDS.

\(^1\)inputs from the GTDA Project Summary and GTDA Problem Statement
1. INTRODUCTION

1.2 The Thesis

The objective of the thesis is to design an Enterprise System Information Architecture for the integration, quality assurance and dissemination of geothermal data. The GTDA platform is built on Service Oriented Architecture (SOA) principles to provide a loosely coupled integration of proprietary systems and open source components. The scope of the design covers the integration and extension points of systems/modules including but not limited to Authentication, Authorization, Full Text Search, Geospatial Web Services and Content Management.

The scope of the thesis includes the design and implementation of an architectural framework which exposes independent services. This framework provides the basis for architectural scalability and flexibility and development of additional or newer components. This thesis will provide a detailed description of the architecture, design and implementation of the architecture of the GTDA system and also cover the implementation aspects of the services incorporated into the system.

1.3 Contributions

The development of the thesis lead to the following contributions:

- As a part of the thesis an event driven SOA system was designed and implemented capturing most of the essential services critical to the project.

- With a plug and play architecture the development time related to the addition of newer services was brought down significantly.

- By adhering to a common set of protocols and exposure endpoints such as REST the architecture becomes flexible to change.

- We were able to rapidly prototype and add extension services in our development code base and achieve parallel and independent module development, a side effect of developing a loosely coupled architecture.

- By reusing existing frameworks there was a significant reduction in our code base and hence development time.
1.4 Layout of the Document

From an implementation perspective several independent and reusable modules were developed and an automated pipeline consuming several of the exposed services in these modules was thoroughly developed.

1.4 Layout of the Document

The rest of the thesis is organized as follows:

- Chapter 2 gives an overview of SOA. Since we make heavy use of Enterprise Integration Patterns (EIP) a description of few EIP’s are also provided to provide the required context.

- Chapter 3 outlines the high level requirements of the system

- Chapter 4 explains the architectural decisions and provides a background of the technology decisions for the GTDA platform.

- Chapter 5 gives a detailed overview of the design decisions for the modules, their packaging and the underlying database schemas.

- Chapter 6 provides the details of the implementation of the system.

- Chapter 7 presents a discussion of the results of the properties of the developed architecture.

- Chapter 8 presents the conclusion of the thesis.

- Chapter 9 explains additional aspects of the project related to project planning etc.
1. INTRODUCTION
2

Background

This chapter gives an introduction to the concepts and ideas which lead to the formation of the thesis. It later gives a brief overview of integration patterns which are central to the implementation of the thesis.

2.1 Service Oriented Architecture

SOA. or Service Oriented Architecture is a design principle and open to many interpretations and implementations. SOA provides design principles and methods for systems development and integration to build loosely coupled systems composed from individual and inter-operable services. A SOA infrastructure then allows different applications to exchange data with one other usually with the help of an underlying messaging framework. By design, SOA is loosely coupled and thus provides enterprises better flexibility in building applications and business process in an agile manner.

2.1.1 Enterprise Service Bus

ESB. or Enterprise Service Bus is a conceptual model for the design and implementation of the communication between different services in SOA. It is a message/event oriented architecture that uses messages as the backbone of integration. The ESB is in charge of routing, filtering, publication and subscription and in some implementations the monitoring of the messaging system. All applications or services communicate via the ESB using protocols supported by the ESB.
2. BACKGROUND

2.1.2 REST

REST, or Representational State Transfer is an architectural style for building loosely coupled systems. A RESTful web service is implemented using HTTP and the principles of REST. It is a collection of resources with three defined aspects:

1. The base URI for the Web Service.
2. The Internet Media Type of the data supported by the Web Service (eg. XML, JSON etc.)
3. The set of operations supported by the web service using HTTP methods (e.g. POST, GET, PUT or DELETE)

REST and JSON are gaining popularity in today’s web-based platforms and applications[1].

2.2 Enterprise Integration Patterns

Enterprise Integration Patterns[1] are a collection of 65 integration patterns described in Enterprise Integration Patterns[2]. These patterns help solve system and data integration problems to develop robust integration solutions and are relevant in both the SOA and ESB space. We look below at a few patterns that we have incorporated into our application design, which will be useful for understanding the design and implementation of the system.

2.2.1 Message

A message consists of headers and a payload. The headers contain data that is relevant to the messaging system, like the return address or correlation ID (which uniquely identifies the message). The payload contains the actual data that is to be accessed or processed by the receiver as shown in figure 2.1.

2.2.2 Message Channel

Producers send Messages to a channel and consumers receive messages from a channel as shown in figure 2.2. A message channel can either follow point-to-point or publish-subscribe semantics.

2.2 Enterprise Integration Patterns

Figure 2.1: **Message** - An eip pattern described at http://www.eaipatterns.com/Message.html

Figure 2.2: **Message Channel** - A pattern described at http://www.eaipatterns.com/MessageChannel.html.
2.2.3 Message Endpoints

Figure 2.3: Message Endpoint - A pattern described at http://www.eaipatterns.com/MessageEndpoint.html.

Messaging Endpoints are the connections between functional services and the messaging framework. The message endpoints handle messages and are mapped to message channels.

2.2.4 Message Translator

Figure 2.4: Transformer - A pattern described at http://www.eaipatterns.com/MessageTranslator.html.

A Message Translator or transformer is used to transform the payload of a message from one format to another. It can also be used to add, modify or remove message headers.

2.2.5 Filter

A Message Filter determines whether a message can be sent to an output channel. It may check for a payload content type, a property value, the presence of a header etc. to determine whether to drop a message or send it to the output channel.
2.2 Enterprise Integration Patterns

Figure 2.5: Filter - A pattern described at http://www.eaipatterns.com/Filter.html.

2.2.6 Content Enricher

Figure 2.6: Header Enricher - A pattern described at http://www.eaipatterns.com/DataEnricher.html.

A Content Enricher enhances a message with more information than was provided by the target system. eg. Adding headers to a message.

2.2.7 Router

Figure 2.7: Router - A pattern described at http://www.eaipatterns.com/MessageRouter.html.

A Message Router determines which channel should receive a channel based on
2. BACKGROUND

metadata or data in payload or header of the message.

2.2.8 Splitter

A Message Splitter splits a message on its input channel into multiple messages and then sends each of these messages onto its output channel.

2.2.9 Aggregator

A Message Aggregator receives multiple messages and combines them into a single message.

2.2.10 Service Activator

A Service Activator connects a service instance to the messaging system. A message received on the input channel invokes an operation on the service which sends the results to an outbound channel.
2.2 Enterprise Integration Patterns

Figure 2.10: Service Activator - A pattern described at http://www.eaipatterns.com/MessagingAdapter.html.

2.2.11 Channel Adapter

Figure 2.11: Channel Adapter - A pattern described at http://www.eaipatterns.com/ChannelAdapter.html.

A Channel Adapter may be inbound or outbound and connects a messaging system to another system or transport (eg. File, http, jms, ftp, jdbc etc.)
3

Requirements

This chapter gives a brief introduction into the project requirements and elaborates on a few details relevant to the thesis.\footnote{with inputs from the GTDA\_Software\_Requirements\_Specification.pdf}

3.1 Project Requirements

3.2 Overview

A high level view of the GTDA system is presented in figure 3.1. An overview of the actors and the use cases is provided in the following sections. As mentioned in the introduction 1.2 the scope of this thesis is to help implement the underlying framework for this system in agreement with SOA principles. The use cases and actors then interact with the services exposed by the underlying architectural framework.

It is in this context that an automated publication pipeline, introduced in the design chapter in section 5.1.2 incorporates the consumption of services (or automatization of use cases) without the need for external actors when possible. The architecture then provides extension points for hooking in manual interactions with the system as described in later chapters. There were certain external requirements as well such as to make the whole system modular and reusable and we expect changes to the original requirements because the users may better express their needs when they have a working system.
3. REQUIREMENTS

![Figure 3.1: Context Diagram - The Context of the GTDA application.](image-url)
3.3 Use Cases

The GTDA system supports and facilitates the uploading, integration and submission of geothermal data. The system checks the integrity of data and performs quality assurance wherever and whenever possible. For those scenarios where the data cannot be automatically validated then the system will allow the owners of the data to process and analyze the data and perform manual quality assurance. The system also handles the submission of meta-data records into the NGDS system and transmission of geothermal data. In brief the system supports the following feature groups:

- Data Input
- Data Browsing
- Quality Assurance
- Data Submission
- System Administration

3.3 Use Cases

This section provides a very high level view of the actors and use cases of the system.

- Actors

1. **NGDS** performs query and retrieval operation on the GTDA system.
2. **System Administrator** performs maintenance tasks of the system including administering users, user roles and data.
3. **Data Submitter** uses the system to input the geothermal data and can view and browse data in the system.
4. **Geothermal Analyst** uses the system to browse, view and export geothermal data.
5. **Data Owner** owns the geothermal data input by the Data Submitter and uses the system to analyze and determine the quality of the input data.

- Use Cases
3. REQUIREMENTS

Figure 3.2: Use Case - High Level use cases of the system.
1. **Import Data Files** The system maintains a Data Import Profile (DIP) for each of the partners associated with NGDS. This DIP has information pertaining to the type, structure for each file which helps the system identify the imported data files and how to handle each data import. This is handled by the data submitter.

2. **Manually Input Data** The system allows manual addition of entries to already existing data catalog entries in the system.

3. **Browse Data Catalog** Once the data has been imported into the system it is possible for users of the system to view and browse the data in the system according to their security permissions.

4. **Search Data Catalog** All data input into the system is indexed so that is viewable in search results for the end-user according to their security permissions.

5. **Export Search Results** The data results can be exported or downloaded by the user.

6. **Manual Quality Assurance** When the automatic quality validation fails the system provides manual quality assurance in the form of a workflow so that the Data Owner can maintain and assess the quality of the data input into the system. In addition the Geothermal Analyst can also flag catalog entries for quality checks.

7. **Request Data/Meta-Data** Once the data catalogs are verified for quality checks, then the meta-data can be published to the NGDS. The NGDS will then be able to request these data items using unique identifiers associated with the data records in the meta-data published to the NGDS.

8. **Query Data** The NGDS is able to query the catalog to retrieve data records associated with the meta-data or to verify the linkage of the meta-data associated with the data.

9. **View Import Logs** The administrator of the system is able track the import of data records into the system.

10. **Administer Users** The administrator of the system is able to define groups and users of the system and have a set of roles associated with users. The
3. REQUIREMENTS

roles then have particular permissions associated with them which allow access to different functionalities within the system.

3.4 Automatization

The previous sections mention the high level use cases for the system. There are a significant amount of back-end processes or services which were developed as part of this thesis to enable these use cases. These include importing data files into a file system, detecting the profiles of the various files, publishing files to the data catalog, making these files indexable and hence searchable. The thesis aims to provide the automatization of the above use cases and develop a framework and the infrastructure on which these services are exposed and help drive development of the manual use cases.
This chapter provides an introduction to integration architectural principles and follows it up with a discussion on the technologies available to support this architecture and the project requirements. The chapter then concludes with the decisions on the choice of technology.

4.1 Architecture

Figure 4.1: SOA - ESB and SOA
4. ARCHITECTURE AND TECHNOLOGY

The above figure provides a semantic overview of the architecture we propose to design. Based on SOA principles reusable and extensible components of the system are developed and exposed as services. Some of these services are core internal services whereas others are deployed externally and hence accessible as external services. All internal services are accessed via the integration bus and external services are accessed via their supported protocols on the integration bus. The integration bus itself is deployed within a Web Application Context which has other internal business modules. The database connection configurations are available on the bus and individual services can override them (for specific business needs) and get a connection handle to the database services.

4.2 Services

The architecture proposal and a detailed analysis of the software requirements for the GTDA project lead to the definition of the following high level services and technology requirements.

- **File System Service** to provide the ability to upload, store, retrieve (download), delete files of different formats. Also provide users of the system CRUD\(^1\) operations on folders providing a virtual file management interface.

- **ETL Service** to provide the ability to launch, stop, restart and monitor ETL\(^2\) jobs. Different ETL jobs are configured to work on their respective file formats for ETL operations.

- **Data Access Service** to provide a consistent access to the Database including data-sources, connection pooling and transaction management.

- **Search Service** providing full text search on all documents in the system.

- **Catalog Service** providing CSW enabled and ISO compliant catalog and search capabilities.

- **Web Feature Service** providing WFS compliant capabilities.

---

\(^1\)Create Read Update Delete  
\(^2\)Extract Transform Load
4.3 Technology Survey

- **Data Analytics** on quality and metrics of the resources in the system.

- **Security** for authentication and authorization with the added capabilities of permissions. It may be useful to extend it to provide SSO services.

- **Data and System Integration technology** for connecting the different services.

- **Rich User Interface** to provide an advanced experience to the end-user.

### 4.3 Technology Survey

**Technology and Project Requirements.** In order to create a match between the SRS, SDS and the technologies required to support the proposed architecture, a technology survey was initiated to explore various kinds of tools, technologies and frameworks to address and solve the project requirements. This section explores the various technologies considered, experimented with and evaluated by giving a brief background of the project requirement. It finally concludes by validating the decisions made in the choice of the technology, tool or framework.

#### 4.3.1 Data Integration

Data and System Integration is at the core of our proposed architecture. The framework must help various services communicate with each other via supported protocols, provide tools to configure and consume services and provide transaction support.

**Apache Camel** is a powerful open source integration platform based on EIP’s. It supports talking to many different transports (HTTP, JMS, File etc.) with Camel components via URI’s. Routing and mediation rules can be configured via Java DSL or Spring based XML configuration. Apache Camel is easy to set up and provides good support with spring. Camel is designed to run inside an ESB like Apache ServiceMix or a pure JMS environment like Apache ActiveMQ.

---

1. Single Sign On
4. ARCHITECTURE AND TECHNOLOGY

**Mule** is a lightweight Java based ESB and integration platform that can connect to various protocols. Since it is lightweight it is highly scalable. Mule provides orchestration facilities with the Mule Studio which can create flows very quickly. For each flow data is received or send from configured endpoints to components or services as chosen and mediated upon by Mule (eg. routing, transformation etc.).

**Spring Integration** extends the core Spring programming model to support the well-known EIP’s. It enables lightweight messaging within Spring-based applications and supports integration with external systems via declarative adapters. These adapters provide a higher-level of abstraction over Spring’s support for remoting, messaging, and scheduling. Flows are configured using spring configuration files or developed using the integration-graph in SpringSource Tool Suite.[1]

### 4.3.2 User Interface

A UI is an important part of an application. In order to support a variety of complex user interactivity and provide a seamless user experience we decided to make the application a Rich Internet Application or RIA. A RIA platform would need to have support for data grids, a rich support of customizable widgets, charts and most importantly an extensible API.

**Adobe Flex** is a very popular UI development platform for RIA’s. It provides a rich API for developing user screens, animations, data grids, maps and charts. Mobile and web applications can be developed using a common code base. However due to performance issues such as slow dynamic rendering of complex UI’s and the unclarity of future support on mobile devices for flex (Android but not iOS) especially with alternatives like HTML5 on the horizon made us explore other alternatives.

**Google Web Toolkit** or GWT is an emerging open source platform for RIA development. Most of the code is written in Java and then efficiently compiled into JS by the GWT compiler. GWT provides customizable pre-built widget. The development environment GWT Designer is plugged into Eclipse, a popular IDE for Java,

---

4.3 Technology Survey

and provides tools for Rapid Application Development (RAD). GWT also popularizes the concept of MVP and adds concepts of Activities and Places. In our evaluation we successfully integrated GWT with Spring back-end services using libraries like gwt-sl, overriding bean instantiation and various other libraries. We tested the architecture with and without MVP architecture and also with GWT-RPC. GWT integrates with Google Charts, however it requires a user to be online for use. The visualization libraries are loaded dynamically and then interact with the data to create the charts. This was against our requirements and there were also a few concerns related to the maturity of the platform especially the frequent updates and releases which made us look at other alternatives of GWT.

**SmartGWT** is a GWT based framework and has an extensive, customizable and rich set of widgets for RIA’s. We evaluated both the LGPL version and commercial version of SmartGWT. The LGPL version requires the developer to create their own Datasource and utilizing their various client-data-integration options, we used RestDataSource, adhere to the DSProtocol. The Enterprise version provides a powerful mapping to SmartGWT’s visual components and the backend and also contains charts. However at the time of our evaluation the charts feature was not yet released which was an important requirement.

**ExtGWT** is a GPLv3 and also commercially licensed GWT based framework and has an extensive set of data grid and visualization charts components. However at the period of our evaluation the ExtGWT framework was being rehauled to support new GWT 2.x features.

**ExtJS** is available under the GPLv3 and commercial licence. It is a JavaScript based web framework with a MVC architecture. It has an elaborate set of data grid components.

---

1. Model-View-Presenter
8. Model-View-Controller
4. ARCHITECTURE AND TECHNOLOGY

components, widgets and visually attractive charts. These components and widgets can be integrated with server-side code using REST. The ExtJS framework is mature, well maintained with good charting features.

4.3.3 Search

The GTDA project requires end-users to have full-text search capability for documents uploaded into the file repository, within the catalog and also within maps to provide an aggregated result.

Apache Lucene is an open-source, high-performance, full-featured text search engine library written entirely in Java. It has a simple API which provides access to its powerful features.

Apache Solr is an open-source standalone full-text search server which embeds the Apache Lucene search library for full-text indexing and search. It also has REST-like HTTP/XML and JSON API’s. Its major features of interest to us included rich document handling, geospatial search, full-text search, highlighting amongst others. During our evaluation we installed Solr within a Apache Tomcat instance and developed a simple API to upload rich text documents and index them to make them available for full-text search.

4.3.4 File Repository

The storage system for the GTDA project has management requirements such as creation of folders to manage files, renaming, deletion etc. This should be flexible to include permission settings. A content repository provides access to such features.

Apache JackRabbit is a content repository fully conforming to JCR API. It supports structured and unstructured content, versioning, transactions amongst others. Since the content stored into JackRabbit can any form of data we decided to evaluate it. We installed the JackRabbit repository and created Workspaces and nodes (representing files and folders). Stress tests on JackRabbit revealed a performance degrade

1http://www.sencha.com/products/extjs/examples/
2http://en.wikipedia.org/wiki/Full_text_search
3http://tomcat.apache.org/
when a single node contains references to more than 10k nodes. The documentation for JackRabbit required extensive search and samples were scant.

**Custom Implementation** of a virtual file system would be our implementation of a lightweight api that creates a wrapper for management services over a filesystem. We develop CRUD operations over a filesystem and provide unique references to files using UUIDs storing the relevant metadata associated with that file in a relational database. This virtual filesystem also supports directories (or folders managed by users) with CRUD operations which can contain other directories or files.

### 4.3.5 Geospatial WebServices

**Geoportal** [27] is an open-source product that enables discovery and use of geospatial resources. It helps organizations to manage and publish their geospatial resources to let users discover and connect to those resources. In our evaluation we installed Geoportal and also downloaded the source code and took an overview of the architecture to determine extensibility of the platform. We evaluated the workflow, permissions, metadata standards, database schema amongst other things.

**GeoServer** [28] is an open-source product that implements the WMS standard and conforms to the WFS standard. It allows to display, publish, share and edit spatial information.

### 4.3.6 Security

**Spring Security** [29] is a powerful and highly customizable access control framework. It provides authorization, web request security and service layer and domain object security which can be configured to run with any type of back-end (JDBC, CAS, LDAP etc.).

**Apache Shiro** [30] is an open-source Java Security framework that provides an api for authentication, authorization and permissions. It has a concept of a realm which could be anything from an Active Directory or JDBC back-end which acts as the repository for credentials and provides a fairly thin API layer over these to manage security concepts.
4. ARCHITECTURE AND TECHNOLOGY

Shibboleth[^1] is a standards based open-source software package for web-single sign on. It uses a federated identity based authentication and authorization infrastructure based on SAML. Its components include an Identity Provider (IdP) and Service Provider (SP). The IdP authenticates the user and provides an authentication token to the SP. SP’s rely on IdP’s to assert the identity of a user.

4.4 Decisions

4.4.1 Spring Integration

To cover the aspects of system and data integration we chose the Spring Integration framework since it provides a very lightweight messaging framework built on top of the familiar Spring Framework. It incorporates the same programming paradigms from the Spring framework and hence the learning curve was believed to be short and in fact was. The adapters provided by the Spring Integration covered the requirements from the project specification and the inherent messaging model can be linked to Spring framework’s application events. Rather than looking for a full blown ESB, we decided to use the lightweight spring integration context for messaging, routing and connecting our services.

4.4.2 ExtJS

We really wanted to try out the GWT platform but the plain GWT from Google did not support many widgets. SmartGWT required us to adhere to their protocol. ExtGWT was being revamped to incorporate GWT2.x features. Another concern was how to connect GWT developed in Java to an existing JS api, which is possible using JSNI[^1] One of the major requirements was to have a robust and visually appealing chart framework. ExtJS provided just that. Since our services for the UI were being exposed via RESTful spring controllers the UI could be independently developed and connected to our services. ExtJS also has its own MVC application design pattern which is quite useful for managing JavaScript code which quickly becomes unmanageable. Owing to the above factors our choice for the UI was to develop our screens using ExtJS.

4.4 Decisions

4.4.3 Solr

Since Solr functions as a standalone web application and it has Java API available to connect to it using a RESTful API using JSON and XML we chose Solr over Lucene which is a library. Also Solr provides all the features which are required by our project and follows the standards proposed in our architecture.

4.4.4 Custom VirtualFileSystem Implementation

Our evaluation of JackRabbit was sluggish mostly because it involved reading a behemoth JCR specification. While we did delve with code and the features seemed promising, they took too much development effort. It was realized soon enough a customized yet extensible virtual file system approach targeted to our requirement could be built rather quickly and hence this approach was preferred.

4.4.5 Geoportal and GeoServer

Since submission of data to the NGDS needed to be performed via OCG web-services\(^1\) as defined by USGIN\(^2\) Geoportal which supports CSW and GeoServer which supports WFS were evaluated for features and extension points and chosen as the standard for catalog and geo-spatial web-services.

4.4.6 Shiro

Spring Security was surprisingly a little bit more complex to understand and implement quickly. We evaluated Shiro and its concepts seemed simple enough to understand for implementation of authentication, authorization and permissions. However, SSO implementation in Shiro requires the use of Cached Sessions. Shibboleth is planned to be revisited in the future for SSO and will primarily deal with the authentication aspect of security and Shiro will deal with the authorization and permissions part.

4.4.7 Spring Batch and Spring Batch Admin

Prior to the beginning of this project certain other ETL services were already being developed using Spring Batch and it provided a robust framework and API for such

\(^1\)http://www.opengeospatial.org/standards/common
\(^2\)http://usgin.org/
services. We extended the Spring Batch Admin, an open-source web application from Spring, to connect to the back-end ETL services to provide a management layer over these services. The management layer provides tasks such as launching, restarting, terminating and monitoring various ETL services. The interesting aspect of using Spring Batch Admin is that it registers the ETL services and exposes them using REST endpoints with JSON, which is the standard of communication in our proposed architecture.

4.4.8 PostgreSQL and PostGIS

The PostgreSQL\(^1\) is an open-source object relational database with a strong reputation and was the choice for our database. PostGIS\(^2\) was chosen to add support for geographic objects to PostgreSQL.

\(^{1}\)http://www.postgresql.org
\(^{2}\)http://postgis.refractions.net/
5

Design

In continuance with the architectural design proposed in 4.1 this chapter expands further on the design of modules based on the services described in 4.2. The chapter first elaborates on the architecture, then on the design of the modules, the database schemas and finally concludes with certain important design decisions made.

5.1 Design

5.1.1 Application.

![SOA Application Design](image)

**Figure 5.1: SOA Application Design** - Modularized components of the architecture

As displayed in figure 5.1 the architecture consists of two web modules, the Server
5. DESIGN

Web Module and the ETL Batch Admin Module. All other modules such as geoportal, search, geoserver exist as independent services. All internal and external services are exposed on and can be consumed via the Message Bus. External services have RESTful endpoints that enable them to be consumed independently of the message bus if required. The Message Bus also incorporates certain Business Wrapper Services which implement the business logic of the GTDA application by consuming the services as required. The Message Bus also tracks the flow of information as it arrives and moves out of the system.

5.1.2 Publication Pipeline.

One of the objectives of the thesis was to develop automated publication pipeline which simulates the business process flow of the life-cycle of a document which is input into the GTDA system. The current publication pipeline has been designed and implemented to be automated and various quality checkpoints will be pushed into place in order to make this integrate with a manual workflow. The automated publication pipeline is described as below:

1. Upload a File. A file is uploaded into the system either via a web form or as part of an ETL service. This file is then stored into the file system.

2. Detect Files. A file uploaded in the system is then detected according to the various data profiles and corresponding formats as explained in the Import Data Profile Use case in the requirements chapter 3.3. A data profile can be one of as listed below.

   (a) Bibliography is an index of publications maintained by the various partners. If a file is detected as a bibliography then the corresponding ETL Service is triggered to load data (the publications) into the corresponding table in the database.

   (b) Publication is the actual file referred to in the bibliography 2a. Currently these files are paper only and universities scan and OCR them and in this system they need to be indexed and linked to the bibliographic record. Thus if a file is detected as a publication from a particular source (partner) then a verification check is done to determine if the publication exists in the
bibliography associated from that source. If such a verification succeeds then it proceeds to index the publication for searching.3

(c) Data File is a file which is not a publication or a bibliography and can be an image file, document, database dump file or any other acceptable data file format. When such a file is detected then Quality Checks are enforced and then it proceeds to be published to geoportal 4 if such a file is to be published to the catalog.

3. Publish to SOLR. Publish the file to SOLR for indexing and full text search.

4. Publish to Geoportal. Create an ISO compliant meta-data for the file and publish it to Geoportal.

5.1.3 ETL.

We identified over twenty ETL services and designed them as Spring Batch jobs configuring and designing the interfaces as described in the reference 1. For example consider an ETL Service to upload an Excel file into a database. To implement this using Spring Batch we need to implement the ItemReader interface 2 to read items from the excel file and an ItemWriter interface 3 to write the read items into the database. We then use the spring xml configurations to wire up these components together.

Listing 5.1: Reading Publications from an Excel file

```java
public class PublicationFileReader implements ItemReader<Publication> {
    private String path = "";
    private String source = ""; // set the partner profile info

    private BufferedReader br = null;
    private String currentLine = null;

    public void initialize() {
        // initialize buffered reader
        br = new BufferedReader(new FileReader(path));
        currentLine = br.readLine();
        // handle exceptions
    }
}
```

2 http://static.springsource.org/spring-batch/apidocs/org/springframework/batch/item/ItemReader.html
3 http://static.springsource.org/spring-batch/apidocs/org/springframework/batch/item/ItemWriter.html
5. DESIGN

```java
@Override
public Publication read() throws Exception, UnexpectedInputException, ParseException {
    if (br == null) return null;
    Publication p = new Publication();
    // read from current line and set values to the publication object
    return p;
}
```

Listing 5.2: Publication writing to a database

```java
public class PublicationWriter implements ItemWriter<Publication> {
    private PublicationDao publicationDao;

    @Override
    public void write(List<? extends Publication> publications) throws Exception {
        for (Publication p : publications) {
            publicationDao.save(p);
        }
    }
}
```

Listing 5.3: Wiring the Spring Batch components

```xml
<bean id="publicationGRCReader" class="com.siemens.scr.gtda.etl.reader.PublicationFileReader" init-method="initialize" scope="step">
    <property name="path" value="#{{jobParameters[ 'path' ]}}" />
    <property name="source" value="#{{jobParameters[ 'source' ]}}" />
</bean>

<bean id="publicationGRCWriter" scope="step" class="com.siemens.scr.gtda.etl.writer.PublicationWriter" >
    <property name="publicationDao" ref="publicationDao"/>
</bean>

<batch:job id="publicationJobGRC">
    <batch:step id="importPublicationGRCStep" parent="simpleStep">
        <batch:tasklet transaction-manager="transactionManager">
            <batch:chunk reader="publicationGRCReader" writer="publicationGRCWriter" commit-interval="200"/>
        </batch:tasklet>
    </batch:step>
</batch:job>
```
5.2 Modules

All the modules in the GTDA projects are designed for reusability, flexibility and extensibility. There is a clear separation of concerns between the modules and hence only the classes and data entities specific to a module are referenced or packaged within that module. Some modules are designed to be independent based on their functionality whereas some modules are referenced by others to consume their services. In essence, unless otherwise mentioned, a module will generally consist of the following packages and files.

1. **Controller.** This package contains the RESTful Controllers to expose the module over HTTP with REST.

2. **Services.** This package exposes the interfaces of the implemented business services specific to that module.

3. **DAO.** This package contains the Data Access Services and Data Entities representing the data models specific for that modules operation.

4. **Spring Configuration File.** The spring context wiring up the components within that module.

![Figure 5.2: A generic Module - General packages and files layout in a module](image)
5. DESIGN

5. Maven pom.xml This file consists of the maven\(^1\) dependencies of this module on other libraries and modules.

5.2.1 Persistence

The persistence module represents the core data model for data specific to the GTDA project and is reusable with other modules. It provides the configuration of the data-source, connection pooling, transaction managers, jpa adapter and the postgis dialect. We use JPA\(^2\) to manage mappings between relational data and java objects and PostGIS\(^3\) to add support for geographical objects to the PostgreSQL database. In future iterations of the project we will consider exposing the JDBC connections as a JNDI service on the integration bus.

```
<!-- Datasource and Connection Pooling configuration -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="$ {jdbc.url}" />
    <property name="username" value="$ {jdbc.username}" />
    <property name="password" value="$ {jdbc.password}" />
    <property name="maxActive" value="$ {jdbc.maxActive}" />
    <property name="maxIdle" value="$ {jdbc.maxIdle}" />
</bean>

<!-- The transaction manager controls transactions with the database. -->
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<!-- jpa -->
<bean id="jpaAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="database" value="POSTGRESQL" />
    <property name="showSql" value="true" />
</bean>

<!-- entity manager -->
<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="jpaVendorAdapter" ref="jpaAdapter" />
    <property name="persistenceUnitName" value="gtdaPU" />
    <property name="jpaProperties" />
```

\(^1\)http://maven.apache.org/
\(^2\)http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html
\(^3\)http://postgis.refractions.net/
5.2 Modules

26
<props>
  <prop key="hibernate.dialect">org.hibernatespatial.postgis.PostgisDialect</prop>
</props>

28
</props>
</bean>

5.2.2 File System

The File System service provides an abstraction of an actual file system, allowing users to manage their respective folders and files. It utilizes Java I/O library to manage disk access, and spring jdbc and jpa to maintain a file index in the database as explained in the filesystem schema[5.4.1]. In addition we provide a REST enabled controller for client (UI) calls to the underlying file management services.

Listing 5.5: File System Service

```java
class FileSystemService {
  public Directory createDirectory(UUID parent, String name);
  public Directory retrieveParentForDirectory(UUID uuid);
  public Directory retrieveParentForFile(UUID fileUuid);
  public boolean removeDir(UUID diruuid);
  public List<Directory> retrieveSubDirectories(UUID parent);
  public Directory retrieveDir(UUID uuid);
  public UUID storeFile(FileInfo file, InputStream content);
  public FileInfo retrieveFile(UUID id);
  public List<FileInfo> retrieveFiles(UUID parent);
  public boolean removeFile(UUID id);
  public boolean moveFile(UUID fileuuid, UUID folderuuid);
  public boolean moveDirectory(UUID srcfolder, UUID dstfolder);
  public UUID storeFile(FileInfo fileInfo);
  public Directory findDirByName(String dirName);
  public Directory renameDir(UUID uuid, String name);
}
```

5.2.3 File Detection

The GTDA system allows users of the system to upload files in various (repeating) formats or data profile. It is useful for the system (for quality checks) to determine the data profile of the files as and when it can and thus avoid confusion for the end-user from a system usability point of view. When the data profile cannot be determined it will be routed to the appropriate authority for detection of its data profile. In the

---

1Input/Output
5. DESIGN

GTDA project a Data Profile consists of the Group it belongs to and whether it is a Publication, Bibliography or any other Data File. We provide a simple interface called FileDetector[5.6] and utilize the Factory Pattern[4] to determine which Detector to use based on the file’s http content type which is determined using the Apache tika[32] library. We utilizing Spring’s xml configurations to achieve this pattern as shown in listing[5.7] and listing[5.8] which retrieves the appropriate detector based on the mimetype.

Listing 5.6: File Detector

```java
public interface FileDetector {
    public String detectFileType(String fileUrl);
}
```

Listing 5.7: File Detection Factory

```xml
<bean id="textDetector" class="com.siemens.scr.gtda.filedetection.detector.Detectors.TextFileDetector"/>
<bean id="msAccessDetector" class="com.siemens.scr.gtda.filedetection.detector.dectors.MsAccessFileDetector"/>
<bean id="excelDetector" class="com.siemens.scr.gtda.filedetection.detector.ExcelFileDetector"/>
<bean id="xlsxDetector" class="com.siemens.scr.gtda.filedetection.detector.ExcelFileDetector"/>
<bean id="xmlDetector" class="com.siemens.scr.gtda.filedetection.detector.XmlFileDetector"/>
<bean id="csvDetector" class="com.siemens.scr.gtda.filedetection.detector.CsvFileDetector"/>
</bean>

<!-- The factory -->
<bean id="fileDetectorFactory" class="org.springframework.beans.factory.config.ServiceLocatorFactoryBean">
    <property name="serviceLocatorInterface" value="com.siemens.scr.gtda.filedetection.detector.FileDetectorFactory"/>
</bean>
</bean>
```

Listing 5.8: File Detector Factory based on mime type

```java
String mimeType = ... // get mimeType using tika libraries
fileDetector = getFileDetectorFactory().getDetector(mimeType)
```

1http://en.wikipedia.org/wiki/Factory_method_pattern
5.2 Modules

5.2.4 Integration

The integration module provides the core of the data integration facilities and is powered by Spring Integration. It references all modules and creates wrapper services (for business specific logic) for access to services exposed by the modules. The integration module also keeps track of all files which are incoming into the system. This module provides the plumbing of services and mediation to form the the publication pipeline.

5.2.5 Server

The server module provides the J2EE Web Deployment Archive to expose the controllers via a Web Interface. This module also hosts the ExtJS UI library and the GTDA application specific developed screens. This module is driven by the Spring MVC library utilizing only the C (controller) part of the MVC library. It is a design decision to expose the Controller via RESTful services and connect to back-end services. The UI components invoke REST calls on these controllers which fetch the requisite data from the associated services. The data format for exchange is JSON. We make heavy use of spring annotations to drive the REST enabled controllers in Spring.

For example take the listing 5.9 which shows a sample RESTful controller

```java
@Controller
@RequestMapping ("/moduleservice")
public class AController {

    @RequestMapping (value = "/doAService" , method = RequestMethod.GET)
    public @ResponseBody String doAService (String queryString , HttpServletRequest request , HttpServletResponse response ) {
        return moduleService.getAService (queryString ;
    }
}
```

In the listing 5.9 the Java annotations convey the following:

1. @Controller. Allows spring classpath scanning to indicate that this class is a Controller

2. **@RequestMapping.** Maps web requests made to `moduleservice` to be handled by this Controller. This can be applied at a class or method level. When used together the associate request for the listing 5.9 would be of the form `/moduleservice/doAService`

3. **@RequestParam.** Binds a method parameter to a webrequest parameter.

### 5.2.6 ETL Services

This module encapsulates all the configurations related to Spring Batch Jobs or ETL services. An ETL Service in most scenarios reads a file in a certain format, does some processing on it and stores it into the relational database. These ETL Services are configured as Spring batch jobs which provides the necessary interfaces and classes to implement such a system very efficiently. We currently have over twenty configured jobs each within their own configuration files. These configurations are then wrapped up or referenced from a single configuration file which provides the extension point for other modules. The database configurations are described in the sub-section GtdaBatch in the Database section 5.4.5 and overview of the design for an elt service was provided in the section 5.1.3.

### 5.2.7 ETL Batch Admin

This module is a custom configured Spring Batch Admin web based module 4.4.7. It overrides the default configurations of Spring Batch Admin to plug into our database back-end. ETL Services can be launched, stopped, inspected and their executions can be monitored using a simple REST enabled web-service using JSON as the data format. It uses the same database configurations as the ETL Service as mentioned in the sub-section GtdaBatch of the Database section 5.4.5.

### 5.2.8 Security

There are two parts to security, authentication and authorization. Using Apache Shiro’s security concepts we designed a schema 5.4.7 and a thin JDBC realm to connect to it. Using shiro filters we are able to intercept every request to check for authentication and using shiro annotations over methods and urls we are able to protect instances

1[^1]

5.2 Modules

and application paths to users with certain roles and permissions. Shiro provides a framework to handle customizable permissions as well. To achieve single-sign-on it is necessary to cache user sessions, however we will be investigating an alternative approach using Shibboleth for authentication and using shiro for authorization. This solution has been deferred to a later project phase.

5.2.9 Workflow

The Workflow module is being developed as a concept to be reusable in the future. The workflow module will be coupled with the Security module described in subsection 5.2.8 to support concepts of permissible viewing and assignment of tasks. The main concept of a workflow is that it is a set of items which progresses through certain states as they are assigned to different users who perform manual operations on an item. Items are uniquely identified by a type which in our case is mostly a File. The manual operations performed by the user will mostly invoke services already defined within the system. These manual operations are required for quality checks which cannot be performed by the automatic publication pipeline system.

5.2.10 Geoportal

The Geoportal server requires Apache Directory Services for authentication and authorization according to roles [user, publisher and administrator]. It also utilizes Lucene for full-text search of content published in its catalog hence requires a folder configuration for indexing. The database configurations are described in the subsection Geoportal in the section Databases 5.4.6.

5.2.11 Solr

The Solr server is deployed within a Apache Tomcat instance and its installation requires folder configuration for indexing and the appropriate jars to be placed in the tomcat endorsed lib folder.
5.3 Packaging of Modules

We follow the J2EE specification on the deployment of J2EE web modules. The deployment environment for these modules is Apache Tomcat. The GTDA application consists of the following J2EE Web Applications described below:

**Server.** The Server Web Module binds the following modules file detection, filesystem, persistence, integration and security. The integration module provides the binding for gluing all the exposed services from other modules. Also the integration module defines the persistence.xml for JPA to create a persistence unit comprising of entities from all the modules. This persistence unit is then utilized by all other modules deployed within the Server module.

**ETL Batch Admin.** The ETL Batch Admin binds the persistence and the ETL services module. The ETL services module defines the persistence.xml for JPA to create a persistence unit comprising the entities from the persistence module.

**Geoportal.** The Geoportal module is a J2EE WAR deployed within the same Tomcat instance with the requisite configurations for authentication and authorization with the configured LDAP server and folders for indexing.

**Solr.** The Solr module is deployed as a J2EE WAR as well with the requisite configurations for indexing.

5.4 Database

**Schemas.** The Database design incorporates various schemas in accordance with achieving a loosely coupled design. Each schema corresponds to a particular module and different schemas may reference each other using constraints as applicable. We describe below the schemas currently available within our system:

5.4.1 FileSystem.

The filesystem schema is the main schema for storing files and managing directories for the application. The schema is designed to incorporate both directories and files,
5.4 Database

as such there are two table Directories and Files. A Directory consists of directories or files, hence the directory table is self referential and the files table has a foreign key to map to its parent directory. The primary identifier for files and directories in the systems is a java.util.UUID object.

5.4.2 Integration.

The integration schema provides the tracking and most relevant information for any file which passes through the system. It provide the capability to track down a file to a particular service and the state in which it currently exists. This is useful for tracking failures and exception scenarios in the system.

1http://download.oracle.com/javase/1,5.0/docs/api/java/util/UUID.html
5.4.3 Core.

The core schema currently consists of only one table for the publications. This schema will be expanded to incorporate different types of data with addition of new tables. The publication table is fairly simple it consists of all the relevant identification or relational attributes for a publication. Since publications are files and are stored in the filesystem an extra column was added to publications to indicate its location in the filesystem. The publication_uuid column is a foreign key referencing uuid column of the file table in schema filesystem.
Figure 5.6: Staging Schema - For datatypes related to geospatial services
5.4.4 Staging.

The staging schema currently is designed to hold data to be presented in the geospatial domain for eg. maps. It consists of tables which represent data mapped to states, counties for representation on a map.

5.4.5 GtdaBatch.

![ETL Schema - Spring Batch tables](image)

Figure 5.7: ETL Schema - Spring Batch tables
The spring batch tables are required by the spring batch framework for execution. Formally a job instance is individually recognized by the job parameters by which it is executed. A job instance can have multiple executions consisting of one or many steps. A job execution context stores the job-level data which is required for a job execution, this is useful to restart jobs from the state they left off. Similarly, the step execution context stores the step-level data which is required for a step execution.

5.4.6 Geoportal.

The Geoportal schema is as designed by the ESRI organization. The gpt_user table contains the full LDAP credentials for users permitted to use the system. The gpt_resource table contains the relevant meta-data for a particular document and the gpt_resource_data contains the full meta-data xml for the document. The gpt_search table helps users to save their searches. The gpt_harvesting_* tables are for when Geoportal is configured to harvest(import) documents from a certain folder.
5. DESIGN

5.4.7 Security.

Using the concepts of Apache Shiro the security schema was designed. We define groups to have many users. There exists many roles in the system and a user can have multiple roles. Hence users and roles have a many-to-many relationship. Each role is associated with certain permissions, conversely users having certain roles have permissions for activities or accessing resources within the system.

5.4.8 workflow

---

Figure 5.9: Security Schema - Security tables based on Apache Shiro concepts

Figure 5.10: Workflow Schema - Workflow concept tables mapped to Users table
The workflow schema 5.10 is a concept schema and is currently under development. The schema models the concept of a workflow consisting of items uniquely identified by a UUID and of a certain type. These items exist in a certain state and are assigned to a user described in the security schema 5.4.7. The current states of an item are NEW, ASSIGNED, FAILED and COMPLETED.

## 5.5 Decisions

### 5.5.1 UUID

To maintain unique references throughout the system, the UUID was chosen to uniquely identify resources, files etc. as they flowed through the system and also to publish resources to the outside world.

### 5.5.2 Integration Wrapper Services

All integration wrapper services business services wrapped around actual services used in the integration module implement the interface in the listing below. This implies that as messages route through the system any integration service which requires a reference to a particular file or resource will be able to acquire it using the UUID 5.5.1.

```java
public interface IntegrationService {
    @Transactional
    public Object service(UUID uuid);
}
```

### 5.5.3 Channels

In our investigation of Spring Integration, various types of Message Channels options were tested. For the purpose of the project the Queue Channel implementation was used, as it does not let transaction support span across the channel as different threads perform the operations on both sides of the channel.

---

1. [http://download.oracle.com/javase/1,5.0/docs/api/java/util/UUID.html](http://download.oracle.com/javase/1,5.0/docs/api/java/util/UUID.html)
5. DESIGN

5.5.4 Transaction Boundary.

To maintain transactions across services, it was decided that the Service operations would demarcate the beginning of a transaction boundary, and hence from 5.5.3 every service would have its own dedicated input queue channel.

5.5.5 Persistence Units.

A JPA persistence unit is a logical grouping of user defined entity classes with related settings. As described in packaging of the modules, the individual persistence units (the persistence.xml files) group the entities required by the different packages. For the Server WAR it is the Integration module and for the ETL Batch Admin it is the ETL Services module.
6

Implementation

This chapter describes the implementation decisions for the components and modules and the integration of the modules. The implementation is divided into three further sections describing the integration of the modules within the Server Web module, the implementation of Security and the implementation of the ETL Batch admin and ETL services module. This chapter does not go into the details about the installation of the Geoportal and Solr modules but rather the implementation of the services used to connect with their exposed services.

6.1 The Chain Pattern

The core framework for the GTDA project is the data and system integration platform for which Spring Integration was chosen. We make inherent use of Enterprise Integration Patterns 2.2 provided by Spring Integration to create flows to implement the publication pipeline design 5.1.2. In the first implementation of our design we utilized

![Message Handler Chain](image)

**Figure 6.1: Message Handler Chain** - Spring Integration's Message Handler Chain helps groups components as a chain
6. IMPLEMENTATION

the basic EIP’s i.e receive a message on a channel, process it and then push it back into a channel. We did expect an overhead for the underlying messaging framework provided by Spring Integration but the overhead was pretty large because of the number of components and channels. We then decided to use the Spring Integration’s Message Handler Chain. This enables us to group components together to form a service chain. We use this approach in our implementation. A message arrives on a particular chain and the messaging overhead was decreased drastically.

6.2 Integration of Modules

Server. Web Module consists of the modules as mentioned in section 5.3. This package addresses the publication pipeline design (section 5.1.2). The implementation is discussed below.

6.2.1 Upload a File

File Upload is a Controller which utilizes the File System Service (section 5.2.2). It exposes two RESTful interfaces or methods using Spring annotations. When a file is uploaded to the filesystem its UUID is generated. A file uploaded to the system via any of these methods is tracked using a pointer to the file (the UUID) and the service and the status that it has progressed through is maintained in the integration schema (section 5.4.2).

1. Upload. This method provides the implementation of uploading a file via a web form. The file is retrieved from the HttpRequest and stored to disk in the specified directory using the filesystem service. Every group has a default workspace or directory within the filesystem and hence if no directory is specified by the user it is uploaded to the default directory. A new UUID is generated for the file being stored.

```
Listing 6.1: Upload

@RequestMapping(value = "/upload", method = RequestMethod.POST)
@ResponseBody
public String upload(
    @RequestParam("file") MultipartFile multipartFile,
    @RequestParam("group") String group,
    @RequestParam("parent") UUID parentDirectory,
)
```
6.2 Integration of Modules

2. Reference. This method provides the implementation of referencing a file located external to the filesystem storage and linking it with the filesystem. In this scenario a new UUID is generated for the file and a reference link (the file URL) is attached to the UUID pointing to its actual physical location. This occurs in some scenarios, like an ETL service, where the file has already been detected or known and hence the data profile for the file can be provided.

```java
@RequestMapping ( value = "/reference" , method = RequestMethod.GET)
@ResponseBody
public String storeReference ( HttpServletResponse response ,
    @RequestParam ( "group" ) String group ,
    @RequestParam ( "profile" ) String profile )
{
    . . .
}
```

6.2.2 Fetch Uploaded files

![Diagram of JDBC Inbound Adapter](image.png)

**Figure 6.2: JDBC Inbound Adapter** - Configured to poll on a database table and push messages to a channel

After the files have been uploaded into the file system storage, we require to fetch these files for detection purposes as described in the publication pipeline. We need to fetch only those files which have recently been uploaded into the system. Using the JDBC inbound adapter we can quickly configure a query with the necessary conditions and trigger it with the use of a poller to fetch a list of files. We configure the JDBC inbound adapter to do an update query on the returned list of files, which we do so, to clearly demarcate detected and undetected files. These selected files are then dumped
6. IMPLEMENTATION

into an output channel. This output channel serves as the input for the chain described in subsection 6.2.3. This configuration can be seen in the listing 6.3. The channel in which to dump the data is the filePoller in line 1, the select query configured spans lines 2-5 and the update query spans lines 5-13. The poller is configured to poll at a rate of every 10 seconds, picking up 200 messages per poll and handling the select and update queries within the same transaction.

Listing 6.3: JDBC Inbound Adapter

```xml
<int-jdbc:inbound-channel-adapter channel="filePoller"
    query="select * from integration.routing_table where
            service_status = 'UPLOADED' and service = 'FILESYSTEM'
            and tracking_uuid is not null"
    data-source="dataSource"
    update="update integration.routing_table set
            service = 'FILEDETECTION',
            service_status =
            case
                when (data_profile is null )
                    then 'DETECTING'
                else 'DETECTED'
            end
            where id in (:id)"/>

<int:poller fixed-rate="10000" error-channel="errorChannel"
    id="tablePoller" default="true" max-messages-per-poll="200"/>
</int:poller>
<int:transactional transaction-manager="transactionManager"/>
<int:channel id="filePoller">
    <int:queue capacity="200"/>
</int:channel>
```

6.2.3 Split the List of Messages and Filter Detected and Undetected Files

The channel filePoller contains a list of messages (list of files). We need to work on individual files and hence split this list into individual messages as shown in the listing below. We utilize the concept of Content Enrichers or Spring Integration’s Header Enricher to add header values to a message which we use to decorate the messages with a header nextService. We assume that messages arriving on the filePoller channel are to be detected and apply the enrichment to the message. Since no reference bean is provided the splitter component correctly assumes the message that it receives from the Header Enricher is of type java.util.Collection and applies the default splitting logic.
6.2 Integration of Modules

Figure 6.3: Chain - A Chain configured to enrich, split, filter and enrich to the collection, incorporating each individual element into a Message and sending it to the next component.

As mentioned in 6.2.2, the detected files have a data_profile whereas the undetected files do not. Therefore we introduce the Filter which checks whether the payload of the message contains has a data_profile or not using a SpEL expression as shown in line 8. The files which have a data_profile and are hence detected are sent to the next component, and the others are discarded and sent to the transformToUUID channel. Hence in this scenario we utilize a Splitter as a simple Router. In continuation of the publication pipeline the next component is a Header Enricher which decorates the messages consisting of detected files to be routed for publication to SOLR. The different service labels are defined in the subsection 6.2.5. Thus it can be seen that Messages from the chain are sent to the output channel transformToUUID in both cases (of detected or undetected files) but with a different value for the header of nextService.

Listing 6.4: Splitter

```xml
<int:chain input-channel="filesPoller" output-channel="transformToUUID">  
  <!-- assume files are to be detected -->  
  <int:header name="nextService" value="FILEDETECTION" overwrite="true"/>
</int:header>  
<int:splitter/>
  <int:filter expression="null != payload.data_profile" discard-channel="transformToUUID"/>
<int:header name="nextService" value="SOLR" overwrite="true"/>
</int:header>
```

1http://static.springsource.org/spring/docs/3.0.x/reference/expressions.html
6. IMPLEMENTATION

6.2.4 Transform the message

![Diagram: Transform To UUID - Filter Detected and Undetected Files]

We reuse this component frequently within the publication pipeline. All message payloads are transformed to UUID’s so that the next services in the pipeline can identify the resources associated with that messages as described in the section 5.5.2. The custom `TransformToUUID` class transforms a message payload (a Collection or a integration.routingInfo entity) to a UUID. The headers for the message are retained and the message is sent to the `routeToNextService` channel.

```xml
<!-- transformer to get the tracking_uuid
    (pass routingInfo objects / rows from integration.routing_table) -->
<int:channel id="transformToUUID"/>
<int:queue capacity="200"/>
</int:channel>
<int:transformer input-channel="transformToUUID"
    output-channel="routeToNextService" method="transform">
    <bean class="com.siemens.scr.gtda.integration.transformer.TransformToUUID"/>
</int:transformer>
```

6.2.5 Route the Messages to the appropriate Services

We utilize the header-router from Spring Integration to implement routing based on the header values in a message. This is also a reusable component and all messages are sent to the `routeToNextService` channel for routing to the appropriate services, by sending the message to the channel for that service. This is done by doing a simple
6.2 Integration of Modules

Figure 6.5: Header Router - Route Messages
check against the value of the header `nextService` as in line 4 in the listing below and the corresponding mapping for that value.

```xml
Listing 6.6: Router

1 <int:channel id="routeToNextService">
   <int:queue capacity="200"/>

3 </int:channel>
   <int:header value="router.header.name="nextService"
   input=channel="routeToNextService">

5 <int:mapping value="FILEDETECTION" channel="fileDetection"/>

7 <int:mapping value="VALIDATION" channel="validatePublications"/>

9 <int:mapping value="ETLJOB" channel="loadBibliography"/>

11 <int:mapping value="GEOPORTAL" channel="publishPublicationsToGeoportal"/>

56

6.2.6 ServiceActivators

Service Activators as the name suggests is a pattern which activates services. In our implementation we have created wrappers around services, mostly for customized business logic, and then using Spring inject the already declared services into the wrapper service. In this sense most of the service activators 5.5.2 utilize actual services and are configured as flows using the EIP’s implemented in Spring Integration. The implementation details are listed below:

6.2.6.1 File Detection

Files whose data profile is not yet known are attempted to be detected by the system, failing which they are pushed to a manual workflow. The messages containing the UUID’s of the files arrive at the fileDetection channel and are fed to the File Detection Chain and then onwards to the transformToUUID channel.
6.2 Integration of Modules

Figure 6.7: File Detection Chain - File Detection Chain

We assume that all messages entering the chain are detected and hence the next service is to be an ETL service. This is the first component of the chain, a header enricher which changes the header nextService to ETLJOB. The second component of the chain is then the service activator for the File Detection Wrapper service which receives the messages with payloads consisting of UUID’s of the files. The wrapper service fetches the file URL’s associated with those UUID’s from the file system service and passes them to the file detection service for detection. The detected result is then fed to the next component. If a file is undetected then the result returned from the File Detection Service is null, and this information is tracked in the integration schema in the routing table. A null message is not sent to the output channel in Spring Integration. The result of the detection from the File Detection Service is of the form 'Group_DataProfile' where Group refers to abbreviations of the data providers and DataProfile is one of Bibliography, Publication or DataFile. We use the filter component after the service with a SpEL expression (line 5) to route the Bibliographies to the transformToUUID channel and the Publications to the the Header Enricher component to change the value of the nextService header to VALIDATION. For DataFiles the current business logic is undefined, but as is evident from this configuration it is easily pluggable into the router configuration and can be branched out to its appropriate channel when its business logic is defined. The enriched messages are then routed to the the channel transform-ToUUID. As can be observed we are reusing existing channels and plugging back into the workflow to route these messages to the next requested service.

Listing 6.7: File Detection

<!— receives the tracking.uuid ; fetches the corresponding path and then proceeds with detection —>
6. IMPLEMENTATION

6.2.6.2 Validate Publications

Messages which contain UUID’s of files which have been detected as publications arrive at the `validatePublications` channel which is the input for the Publication Validation Chain. This Chain determines whether a publication belongs to a bibliography and then sends the message to the `transformToUUID` channel.

The Validator Wrapper Service determines whether a publication belongs to a bibliography. It uses pattern matching on the name of the retrieved file, hence uses the file
system service and uses the publicationDao for looking up the bibliographic reference. If the publication is validated then the referenced UUID is persisted to that publication and the message is sent to the next component, a header enricher, to change the value of the `nextService` header to SOLR.

Listing 6.8: Validate Publications

```xml
<bean id="publicationValidator" class="com.siemens.scr.gtda.integration.services.validator.PublicationValidatorISImpl">
    <property name="fileSystemService" ref="fileSystemService"/>
    <property name="publicationDao" ref="publicationDao"/>
    <property name="routingDao" ref="routingDao"/>
</bean>

<int:channel id="validatePublications">
    <int:queue capacity="200"/>
</int:channel>

<int:chain input="validatePublications" output="routeToNextService">
    <int:service-activator ref="publicationValidator" method="service"/>
    <int:header-enricher>
        <int:header name="nextService" value="SOLR" overwrite="true"/>
    </int:header-enricher>
</int:chain>
```

6.2.6.3 Load Bibliography

Messages which contain UUID’s of files which have been detected as a bibliography arrive at the `loadBibliography` channel which is the input for ETL Launching Service.
6. IMPLEMENTATION

The ETL services are deployed within the ETL Batch Admin Web context and exposed with REST endpoints. We utilize the Spring framework's RestTemplate (lines 12-19, configured with a StringHttpMessageConverter to handle the JSON request and response strings. The string $gtdaJobLaunchURL is replaced with the string http://serverURL:port/spring-admin/jobs/jobName.json?jobParameters and the ETL Wrapper Service utilizes the data profile to figure out the corresponding jobName and the file system service to retrieve the fileURL passed on as one of the jobParameters for that ETL job.

Listing 6.9: Load Bibliography

```xml
<bean id="gtdaJobLaunchService" class="com.siemens.scr.gtda.integration.services.batch.BatchJobsLaunchingServiceISImpl">
    <property name="restTemplate" ref="gtdaJobLaunchRestTemplate"/>
    <property name="routingDao" ref="routingDao"/>
    <property name="gtdaJobLaunchURL" value="${gtdaJobLaunchURL}"/>
</bean>

<bean id="gtdaJobLaunchRestTemplate" class="org.springframework.web.client.RestTemplate">
    <property name="messageConverters">
        <list>
            <bean id="stringMessageConverter" class="org.springframework.http.converter.StringHttpMessageConverter"/>
        </list>
    </property>
</bean>
```

Figure 6.10: Load Bibliography - Load Bibliography
6.2 Integration of Modules

6.2.6.4 SOLR Publisher

Messages which contain UUID’s of publications which are validated arrive at the `publishPublicationsToSolr` channel which is the input for the Solr Wrapper Service. This service utilizes the file system service to fetch the URL for the publication. SOLR deployed within a tomcat container exposes REST endpoints and we developed a services lines 4-8, [6.10] to publish pdf’s, text using this RESTful service to SOLR.

Listing 6.10: SOLR Publisher

```xml
<!-- SOLR SERVICE -->
<int:channel id="publishPublicationsToSolr">
  <int:queue capacity="200"/>
</int:channel>
<bean id="solrService" class="com.siemens.scr.gtda.integration.services.solr.SolrRestService">
  <property name="solrRestURL" value="${solr.resturl}"/>
  <property name="solrURL" value="${solr.url}"/>
</bean>
<bean id="solrPublisher" class="com.siemens.scr.gtda.integration.services.solr.SolrServiceImpl">
  <property name="routingDao" ref="routingDao"/>
  <property name="fileSystemService" ref="fileSystemService"/>
</bean>
```

Figure 6.11: SOLR Publisher - Solr Publisher
6. IMPLEMENTATION

Filtering Publications to be sent to the Catalog Service. The SOLR publisher chain consists of the service activator of the Solr Wrapper Service. Some publications are not required to be sent to the Catalog Service. We therefore use a Filter with a SpEL expression lines 4-6, [6.11] which filters out messages which belong to a particular list lines 8-11, [6.11] and send the rest of the messages to the next component, the header enricher, to change the value of the nextService header to GEOPORTAL.

Listing 6.11: SOLR Publisher Filter

```xml
<util:list id="filterCatalogList">
  <value>OSTI</value>
  <value>STANFORD</value>
</util:list>
<int:chain input-channel="publishPublicationsToSolr" output-channel="transformToUUID">
  <int:service-activator ref="solrPublisher" method="service"/>
  <int:filter_expression="! @filterCatalogList.contains(payload.groupSource )" discard-channel="logChannel"/>
  <int:header-enricher>
    <int:header name="nextService" value="GEOPORTAL" overwrite="true"/>
  </int:header-enricher>
</int:chain>
```

6.2.6.5 GEOPORTAL Publisher

Messages which are to be published to the catalog service provided by Geoportal are
received on the `publishPublicationsToGeoportal` which is the input for the Geoportal Wrapper Service. Geoportal exposes itself via a REST endpoint and we therefore again make use of the Spring REST template (line 26-42, 6.12) which also provides a mechanism to automatically login using basic authentication. Since we post XML data to the REST url, the message converted which is used for this REST template is the `ByteArrayHttpMessageConverter`[1]. The XML data is posted using a template designed from the ISO 19139/19115 Dataset standard. We use this template to push in required data elements using our custom developed ISOPublicationGenerator lines 16-20[6,12] and then post the XML stream to Geoportal. The xml marshalling and unmarshalling is handled using Apache XmlBeans.

### Listing 6.12: Geoportal Publisher Filter

```xml
<!—- GEOPORTAL SERVICE —->
<int:channel id="publishPublicationsToGeoportal">
    <int:queue capacity="200"/>
</int:channel>

<bean id="geoportalPublisher"
    class="com.siemens.scr.gtda.integration.services.geoportal.GeoportalServiceISImpl">
    <property name="geoportalService" ref="geoportalService"/>
    <property name="publicationDao" ref="publicationDao"/>
    <property name="routingDao" ref="routingDao"/>
    <property name="isoTemplateGenerator" ref="ISOTemplateGenerator"/>
</bean>

<bean id="geoportalService"
    class="com.siemens.scr.gtda.integration.services.geoportal.GeoportalRestService">
    <property name="geoportalRestURL" value="${geoportal.resturl}"/>
    <property name="restTemplate" ref="geoportalRestTemplate"/>
</bean>

<bean id="ISOTemplateGenerator"
    class="com.siemens.scr.gtda.integration.services.geoportal.ISOPublicationGenerator">
```

6. IMPLEMENTATION

```xml
<property name="geoportalTemplate" ref="geoportalTemplateFile" />
</bean>
<bean id="geoportalTemplateFile"
class="org.springframework.core.io.ClassPathResource">
<constructor-arg index="0" value="geoportal-iso-19115-template.xml" />
</bean>

<!-- Geoportal Rest configuration -->
<bean id="geoportalRestTemplate"
class="org.springframework.web.client.RestTemplate">
<property name="requestFactory">
<bean
class="com.siemens.scr.gtda.integration.services.geoportal.
BasicAuthenticationCommonsClientHttpRequestFactory">
<constructor-arg name="user" value="${geoportal.user}"/>
<constructor-arg name="password" value="${geoportal.password}"/>
</bean>
</property>
</bean>

<property name="messageConverters">
<list>
<bean id="messageConverter"
class="org.springframework.http.converter.ByteArrayHttpMessageConverter">
</bean>
</list>
</property>

<!-- currently assuming that -->
<int:service-activator input-channel="publishPublicationsToGeoportal"
ref="geoportalPublisher" method="service" output-channel="logChannel"/>
```

6.3 Security

Our current security implementation is based on Shiro. The security module incorporates the security schema design from 5.4.7. We designed a top level extensible interface for access to Shiro security related concepts Users, Roles and Permissions. We extended these shiro concepts to add a dimension for Groups. So in our implementation Groups have Users who can have multiple Roles with certain permissions. The interface is listed below

```
Listing 6.13: Security Service

1 import org.apache.shiro.authc.AuthenticationInfo;
2 import org.apache.shiro.authz.AuthorizationInfo;
3 /**
4 * A Service to wrap around calls to the underlying
5 * Shiro Realm
6 */
```
6.3 Security

```java
public interface SecurityService {
    public AuthenticationInfo getAuthenticationInformation(String userName, String realmName);
    public AuthorizationInfo getAuthorizationInformation(Long userId);
}
```

The implementation for this interface uses an underlying Dao to verify the user’s credentials from a particular realm. A realm\(^\text{[38]}\) is a shiro concept for accessing application specific user security data. Our implementation of the realm uses the database with JDBC calls which expects the stored user credentials to be SHA-256 hashed.

### Listing 6.14: Security Configuration Filter

```xml
<bean id="userDao" class="com.siemens.scr.security.dao.impl.UserDaoImpl"/>
<bean id="securityService" class="com.siemens.scr.security.services.SecurityServiceImpl" scope="singleton">
    <property name="userDao" ref="userDao"/>
</bean>
<bean id="GTDARealm" class="com.siemens.scr.security.services.realm.GTDARealm">
    <property name="name" value="GTDARealm"/>
    <property name="credentialsMatcher" ref="sha256"/>
    <property name="securityService" ref="securityService"/>
</bean>
<bean id="sha256" class="org.apache.shiro.authc.credential.Sha256CredentialsMatcher"/>
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
    <property name="realm" ref="GTDARealm"/>
</bean>
```

Since our application will be packaged and deployed as a WAR\(^\text{[5,3]}\), we need Shiro to intercept Web Requests, hence we make use of a HTTP Filter\(^1\) more precisely a Shiro Filter\(^2\) lines 11-30, 6.16 for intercepting web requests and protecting URLs based on user, roles, permissions or a combination of them all. To be able to use Shiro annotations within a spring context further configuration is required lines 2-10, 6.16. We can then decorate our methods using shiro annotations\(^3\) to provide security at method level as well.

### Listing 6.15: Shiro Filter Configuration protecting REST URLs

```xml
<l!-- Spring AOP auto-proxy creation (required to support Shiro annotations) -->
<bean
```
6. IMPLEMENTATION

```xml
<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"
depends-on="lifecycleBeanPostProcessor"/>

<bean class="org.apache.shiro.spring.security.interceptor.
AuthorizationAttributeSourceAdvisor">
  <property name="securityManager" ref="securityManager"/>
</bean>

<!−− Post processor that automatically invokes init() and destroy() methods −−>

<bean id="lifecycleBeanPostProcessor" class="org.apache.shiro.spring.
LifecycleBeanPostProcessor"/>

<bean id="shiroFilter" class="org.apache.shiro.spring.web.
ShiroFilterFactoryBean">
  <property name="securityManager" ref="securityManager"/>
  <property name="loginUrl" value="/login.html"/>
  <property name="successUrl" value="/index.html"/>
  <property name="unauthorizedUrl" value="/unauthorized"/>
  <property name="filters">
    <util:map>
      <entry key="authc">
        <bean class="org.apache.shiro.web.filter.
authc.PassThruAuthenticationFilter"/>
      </entry>
    </util:map>
  </property>
</bean>

The above configurations are to enable Shiro security but to use them we have to
add the following configuration for the shiro filter to the web.xml of our web application
which is in the server module.

Listing 6.16: Shiro Filter Configuration protecting REST URLs

```xml
<filter>
  <filter-name>shiroFilter</filter-name>
  <filter-class>org.springframework.web.
filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
  <filter-name>shiroFilter</filter-name>
  <url-pattern>/</url-pattern>
</filter-mapping>

```
6.4 ETL Batch Admin

The ETL Batch admin is based on the open-source Spring Batch Admin tool. In order to customize the tool the reference instructions were followed. However in order to override the default properties we had to do the following configurations so that our configuration were loaded first. The injected values for data-source, jobLauncher etc. are then substituted with our configured properties. The configured spring batch jobs in the ETL services module are imported and registered with jobRegistry bean. The ETL Batch Admin then acts as a RESTful JSON webservice and helps to manage the jobs.

Listing 6.17: Overriding Default Configurations of the Spring Batch Admin tool

```xml
<bean id="customProperties" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>file://${app.config}/*.properties</value>
        </list>
    </property>
    <property name="ignoreUnresolvablePlaceholders" value="true" />
    <property name="ignoreResourceNotFound" value="true" />
    <property name="order" value="0" />
</bean>
```
6. IMPLEMENTATION
Evaluation

This chapter evaluates a few key aspects of SOA, coupling in the system, reusability of the system, throughput and an estimate for development effort saved.

7.1 Environment

All experiments described in this chapter were carried out on a machine with the following configurations and software installed:

1. Windows 7, 64 bit machine with 6 GB RAM
2. Apache Tomcat 7
3. Spring 3.0.5
4. Spring Integration 2.0.3
5. Spring Batch 2.1.8
6. Geoportal 1.1
7. GeoServer 2.0.3
8. Apache Directory Services 1.5.7
9. PostgreSql 9.0.4
7. EVALUATION

7.2 Degree of Loose Coupling

Loose coupling is a highly desirable property of a SOA system or any software system. It enables component reuse, parallel development, simplifies unit testing, enables scalable and flexible architectures and also has benefits related to project management. Since our design is focused on implementing a loosely coupled and flexible architecture, we consider the degree of loose coupling an important measurement in the analysis of our architecture. The system is composed of different modules and based on the interactions between modules we show a basic visualization of the coupling which exists between the modules in our system as in figure 7.1.

![Figure 7.1: Module Coupling](image)

Figure 7.1: Module Coupling - A visualization of coupling of modules in the system

In this section we use one of the methodology available to evaluate loose coupling and publish the results. Fenton and Melton\[42] proposed the coupling types as shown in table 7.1 where the order of worst type of coupling to best runs from top to bottom. They then proposed a measure of coupling between modules \( x \) and \( y \) as

\[
M(x, y) = i + \frac{n}{n + 1} \tag{7.1}
\]
7.2 Degree of Loose Coupling

where \( i \) = the measure of the highest level of coupling between module \( x \) and \( y \) and 
\( n \) = the number of interactions between modules \( x \) and \( y \)

The global coupling of a system may be defined as the average of the pairwise coupling of the modules. However, Fenton and Melton propose that the global coupling of a system \( S \) consisting of modules \( D_1, \ldots, D_n \) is given by the median value of the set 
\[ M(D_i, D_j) : 1 \leq i < j \leq n \]
where \( M \) is the measure of pairwise coupling given above. We use the formula \[7.1\] and derive the results for the value of coupling as show in table \[7.2\]

<table>
<thead>
<tr>
<th>Coupling Type</th>
<th>Coupling Level</th>
<th>Modified definition between modules ( x ) and ( y ) from [43]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>5</td>
<td>Component ( x ) refers to the internals of component ( y ), i.e., it changes data or alters a statement in ( y ).</td>
</tr>
<tr>
<td>Common</td>
<td>4</td>
<td>Components ( x ) and ( y ) refer to the same global data.</td>
</tr>
<tr>
<td>Control</td>
<td>3</td>
<td>Component ( x ) passes a control parameter to ( y ).</td>
</tr>
<tr>
<td>Stamp</td>
<td>2</td>
<td>Component ( x ) passes a record type variable as a parameter to ( y ).</td>
</tr>
<tr>
<td>Data</td>
<td>1</td>
<td>Components ( x ) and ( y ) communicate by parameters, each of which is either a single data item or a homogenous structure that does not incorporate a control element.</td>
</tr>
<tr>
<td>No Coupling</td>
<td>0</td>
<td>Components ( x ) and ( y ) have no communication, i.e., are totally independent.</td>
</tr>
</tbody>
</table>

Table 7.1: Coupling Levels - Fenton and Melton Modified Definition for Myers Coupling Level.

In table \[7.2\] the top table shows the values of the pairwise coupling values where
\( i \) is the highest value of the coupling level between a pair of module \( x \) and \( y \) and \( n \) is the number of interactions between those modules. The middle table, displays the evaluated results for the pairwise coupling using the formula \[7.1\] As can be seen from the results in the bottom table in \[7.2\] the coupling of the system is extremely low. The
7. EVALUATION

Table 7.2: Pair Wise Coupling levels from Fenton and Melton Definition table 7.1

<table>
<thead>
<tr>
<th>c=coupling level</th>
<th>file system</th>
<th>persistence</th>
<th>uuid patch</th>
<th>file detection</th>
<th>integration</th>
<th>server</th>
<th>etl services</th>
<th>etl batch admin</th>
<th>solr</th>
<th>geoportal</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=no. of interactions</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>file system</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>persistence</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>uuid patch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>file detection</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>integration</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>server</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>etl services</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>etl batch admin</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>solr</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>geoportal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coupling b/w modules</th>
<th>Mean</th>
<th>0.366</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3: Top : Pair Wise Coupling levels from Fenton and Melton Definition table 7.1
Middle : Pairwise coupling values using formula 7.1
Bottom : Calculation results for coupling of the system.
median value of the pairwise coupling as proposed by Fenton and Melton\cite{42} is 0 or the lowest possible. The average for the same results in a value of 0.366 for the system which is towards the lowest level of coupling value as shown in the table \ref{table:7.1}.

### 7.3 Throughput

The architecture and the implementation design were tested for throughput. We carried out experiments and noted the throughput under different scenarios. Our implementation of Spring Integration using the poller with queue channels causes the processes on each message to be run on separate threads. Hence we created a mock service composed of other services in the system to form one coupled monolithic service and configured it to run with a thread pool to compare its performance with the messaging framework of Spring Integration. We refer back to the implementation of the publication pipeline (section \ref{section:5.1.2}) as the test bed. The scenarios and results from the same are listed below:

- *Using the Mock Coupled Service*
  - A single publication through the publication pipeline using the Mock Coupled Service: On an average over a few runs the processing time for one publication was 3.09 seconds.

![Throughput](image1.png)  
**Figure 7.2:** Throughput of the system using the Mock Service

![Box Plot](image2.png)  
**Figure 7.3:** Box Plot for the time executions through the Mock Service
7. EVALUATION

- Five hundred publications were passed through the publication pipeline. The graph for the throughput is as shown in the figure 7.2 and the box-plot are as shown in figure 7.3. The number of publications processed per second are 2.43.

- Using Spring Integration

  - A single publication through the publication pipeline using the services on the spring integration framework: On an average over a few runs the processing time for one publication was 7.26. seconds.

![Figure 7.4: Throughput of the system using Spring Integration](image)

![Figure 7.5: Box Plot for the time executions using Spring Integration](image)

- Five hundred publications were passed through the publication pipeline. The graph for the throughput is as show in the figure 7.4 and the box-plot is as shown in figure 7.5. The number of publications processed per second are 14.53.

The results for this experiment were surprising. For a single publication through the pipeline the framework was expectedly slow because of the overheads related to messaging. However for a run consisting of several publications through the publication pipeline (which can handle them in parallel) the throughput or the number of publications processed per second is larger than that of the mock coupled service. This shows that the Spring Integration framework is quite robust for multi-threaded processing and our publication pipeline does not suffer from a messaging overhead over a parallel workload. Over a large run of parallel executions the performance of the framework is better than of the multi-threaded mock coupled service.
7.4 Degree of Reusability

Reusability is defined as the number of components that offer reuse in terms of either reusing existing services themselves (internal reuse) or in the number of different ways that they themselves can actually be reused (external reuse) [44].

Taking a look at Figure 7.6, we see that every module other than the uuid patch and file detection, which are very independent and specific modules, is being used by another module within the system. Apart from this internal reusability, five of the total ten modules are exposed via RESTful endpoints, making them available for reuse for any other service external to the system. Hence it can be observed that our architecture has a good balance of internal and external reuse.

7.5 Development Effort

Professional software project management mandates completion of tasks and activities within established time targets. In the software world this is established by the development of reusable components as described in 7.4 or reusing existing libraries for abstract tasks. We used a variety of libraries within our software development for
7. EVALUATION

abstract tasks such as transaction management, connection pooling, object relational mapping amongst many others. An analysis of code coverage of the classes under the following core scenarios was undertaken

- **ETL.** This scenario is that of a simple ETL job of which we have over 20 configured jobs. So this is a repeatable scenario within our code base and hence an analysis of a chosen ETL was taken. Since the ETL services make heavy use of the Spring Batch framework, it makes an interesting scenario.

- **ETL job with REST.** This scenario is that of an ETL service connecting with services exposed by our Integration service. This is also a repeatable scenario within our code base and checks the code coverage of both the ETL Batch admin module and the Server Web Module.

- **The Pipeline.** This covers the automated publication pipeline as described in 5.1.2

The idea is to show the amount of code written vs reusing existing open source libraries for the same task, and hence the corresponding decrease and savings in the development effort. The results of this analysis as shown in tables. 7.4

<table>
<thead>
<tr>
<th>Total Interfaces</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Classes Written</td>
<td>159</td>
</tr>
<tr>
<td>classes written used</td>
<td>Library classes used</td>
</tr>
<tr>
<td>ETL JOB</td>
<td>20</td>
</tr>
<tr>
<td>ETL JOB with REST</td>
<td>11</td>
</tr>
<tr>
<td>Automated Pipeline</td>
<td>60</td>
</tr>
</tbody>
</table>

**Table 7.4:** Top : Total Classes Written  
Bottom : Code Written vs Reusing Libraries

As can be seen from the table, the effort, estimated as a percentage of the classes actually written vs the number of classes reused from different libraries, is minimal ranging from 0.7% to 4.2%. There is some effort involved in learning the API of the different libraries but considering that the library classes can provide reusable code of upto 90% shows a significant savings in development effort.
7.6 Discussion

The above evaluations lead to the following conclusions related to our assessment of our architecture. The architecture is highly loosely coupled as is evident with the lowest coupling levels achieved as described in the table 7.2. The system has a fair degree of reusability as described in the section 7.4 with a high degree of internal module reuse and several modules supporting external RESTful interfaces. The system performs well under a heavy and parallel load execution and the throughput of the system is acceptable. The system uses a considerable amount of frameworks and external libraries and as such the development effort to develop the system was fairly reduced.
7. EVALUATION
8

Conclusions

The conclusion of the thesis and current state of the implementation are provided in section 8.1 followed by future work.

8.1 Conclusions

Event driven SOA is a paradigm in software development which helps develop architectures that are loosely coupled, protocol-independent, location agnostic, coarse grained amongst other characteristics.

This thesis presents the architecture, design and implementation and working of an event driven SOA for services pertaining to the dissemination of geothermal data. This thesis uses Spring Integration to develop the integration bus and relies on the Spring framework for dependency injection and ETL Services. Multiple Services are exposed on the integration bus and can be consumed by other services.

The current implementation supports the uploading of various document formats which are pushed into an automated pipeline on the integration bus which routes and filters the messages to services which process or consume these messages and then push them back on the integration bus. Experiments carried out on the deployed architecture showed that the system was loosely coupled, had a high level of reusable components and the throughput of the system developed was unaffected by the messaging overhead of the underlying integration bus.
8. CONCLUSIONS

8.2 Future Work

The current implementation of the architecture utilizes its own tracking mechanism for documents flowing through the system and service failures. Spring integration provides an implementation of a Message Store \[45\] which provides a strategy for persisting messages. This pattern will be explored further for scenarios involving service and system failures. The implementation of a module, the workflow module, requiring manual intervention for quality checks need to be integrated with the event driven nature of the underlying bus. The way this is achieved is by raising Spring Application Events \[46\] whenever a manual action is performed which are then captured by the integration bus and routed to the appropriate service or event handler.

The security work in this thesis provided a mini-framework to initiate development related to security attributes like authentication, authorization and permissions. The work involving permissions within the system whether at a resource, instance level etc. needs to be expanded upon.

The search interface is to be expanded to include full-text search and geo-spatial search within the same interface. Search results will be accumulated and displayed in a user friendly way on the map as well as a listing of associated documents.

Future work also includes developing or exploring a system to monitor the health of services in the system, exploring a service registry interface and expanding on the geospatial web services capabilities of the system.
Appendix

9.1 Plan

9.1.1 Initial plan

The initial planning covers the design of the architecture and the analysis of the various technologies which were proposed to be used in this project. The figure 9.2 describes in detail the plan of the tasks associated with different aspects of the design and analysis phase.

9.1.2 Development plan

The development plan as show in figure 9.2 gives the detailed planning of the tasks associated with the development of the modules and their associated services. The Services as described in the figure correspond to the modules as follows:

- Server Module File Upload
- File Detection File Detection
- Search Solr Indexer and Search Services
- Geoportal Geoportal Publisher
- File System File System
- Security Authentication and Authorization
- GeoServer Map Services
## 9. APPENDIX

### Figure 9.1: Initial Investigation - Design and Analysis planning

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Predecessors</th>
<th>Resource Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Information Architecture</td>
<td>151 days</td>
<td>Mon 3/7/11</td>
<td>Mon 4/2/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design and Analysis</td>
<td>39 days</td>
<td>Mon 3/7/11</td>
<td>Thu 4/22/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis</td>
<td>12 days</td>
<td>Mon 3/7/11</td>
<td>Tue 3/22/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement Analysis</td>
<td>4 days</td>
<td>Mon 3/7/11</td>
<td>Thu 3/11/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overview of Geospatial software</td>
<td>3 days</td>
<td>Wed 3/10/11</td>
<td>Fri 3/11/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overview of Current technology stack</td>
<td>3 days</td>
<td>Tue 3/15/11</td>
<td>Thu 3/17/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology/Frameworks for POC</td>
<td>3 days</td>
<td>Fri 3/18/11</td>
<td>Tue 3/22/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis of Geoportal</td>
<td>5 days</td>
<td>Wed 3/23/11</td>
<td>Tue 3/29/11</td>
<td></td>
<td>Anshuman</td>
</tr>
<tr>
<td>Geoportal Installation</td>
<td>2 days</td>
<td>Wed 3/23/11</td>
<td>Thu 3/29/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geoportal Architecture analysis</td>
<td>2 days</td>
<td>Fri 3/25/11</td>
<td>Mon 3/28/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geoportal Feature analysis</td>
<td>2 days</td>
<td>Fri 3/25/11</td>
<td>Mon 3/28/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geoportal Database analysis</td>
<td>1 day</td>
<td>Tue 3/30/11</td>
<td>Tue 3/31/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database Schema for ETL</td>
<td>2 days</td>
<td>Wed 3/30/11</td>
<td>Thu 4/1/11</td>
<td></td>
<td>Anshuman</td>
</tr>
<tr>
<td>Analysis of AZU proposed schema</td>
<td>1 day</td>
<td>Wed 3/30/11</td>
<td>Wed 4/1/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development of Schema for Web Logs</td>
<td>1 day</td>
<td>Thu 4/1/11</td>
<td>Thu 4/3/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site Search</td>
<td>2 days</td>
<td>Fri 4/1/11</td>
<td>Mon 4/4/11</td>
<td></td>
<td>Anshuman</td>
</tr>
<tr>
<td>Site installation</td>
<td>1 day</td>
<td>Fri 4/1/11</td>
<td>Fri 4/1/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>File upload to Site</td>
<td>1 day</td>
<td>Mon 4/4/11</td>
<td>Mon 4/4/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spring Integration</td>
<td>6 days</td>
<td>Tue 4/5/11</td>
<td>Tue 4/12/11</td>
<td></td>
<td>Anshuman</td>
</tr>
<tr>
<td>Simple Integration Analysis</td>
<td>3 days</td>
<td>Tue 4/5/11</td>
<td>Thu 4/7/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development of Integration Module</td>
<td>3 days</td>
<td>Fri 4/8/11</td>
<td>Tue 4/12/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration with Site</td>
<td>1 day</td>
<td>Mon 4/11/11</td>
<td>Mon 4/11/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration with Geoportal</td>
<td>1 day</td>
<td>Tue 4/12/11</td>
<td>Tue 4/12/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GWT</td>
<td>6 days</td>
<td>Wed 4/12/11</td>
<td>Wed 4/16/11</td>
<td></td>
<td>Anshuman</td>
</tr>
<tr>
<td>Analysis of gwt apps architecture</td>
<td>3 days</td>
<td>Wed 4/12/11</td>
<td>Fri 4/16/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis of Integration of gwt with spring</td>
<td>3 days</td>
<td>Wed 4/12/11</td>
<td>Wed 4/16/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smart GWT</td>
<td>6 days</td>
<td>Thu 4/21/11</td>
<td>Thu 4/26/11</td>
<td></td>
<td>Anshuman</td>
</tr>
<tr>
<td>Analysis of enterprise license edition</td>
<td>3 days</td>
<td>Thu 4/21/11</td>
<td>Mon 4/25/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis of Enterprise license edition</td>
<td>3 days</td>
<td>Thu 4/21/11</td>
<td>Thu 4/26/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Services Analysis</td>
<td>7 days</td>
<td>Thu 4/21/11</td>
<td>Fri 4/22/11</td>
<td></td>
<td>Anshuman</td>
</tr>
<tr>
<td>Definition of Services</td>
<td>2 days</td>
<td>Thu 4/21/11</td>
<td>Fri 4/22/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First High Level Architecture diagram</td>
<td>1 day</td>
<td>Thu 4/21/11</td>
<td>Thu 4/21/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExtJS and ExtGWT</td>
<td>3 days</td>
<td>Thu 4/21/11</td>
<td>Mon 4/25/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis of tests and licenses</td>
<td>3 days</td>
<td>Thu 4/21/11</td>
<td>Mon 4/26/11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 9.1 Plan

**Figure 9.2: Implementation - Development planning**
9. APPENDIX

- ETL Services File Import

9.1.3 Variance

1. The schema for well logs was designed, however feedback from project partners was not available in time to complete the design for the thesis.

2. The security implementation for authorization and authentication was developed based on Shiro. However there was a late decision for the use of Shibboleth for authentication and its support for single sign on. Since the scope of Shibboleth is too vast the security implementation is currently not fully defined.

3. The workflow management design and implementation requires the security module for permission etc. and therefore it is currently concept based and a prototype has been developed. The design is to be extended further to cover the requirements.

4. The design of the deployment architecture is too premature to be developed given the current security restrictions and information regarding the final deployment area.

9.2 User Screens

1. File System The figure 9.3 shows the user interface of the File System Service described in 5.2.2. The navigation view allows the user to navigate the file system and the main content view shows the detailed view of the folder or file selected in the navigation view.

2. File Upload The figure 9.4 allows a user to upload a particular file to the current directory.

3. Administration The figure 9.5 shows the tracked state of a document in the system. The information includes the tracking identifier of the document, the last service which accessed the document, the status of the service when processing the document in, the time when the document was last accessed and the source to which the document belongs.
9.2 User Screens

Figure 9.3: File System Browser - The user interface for the File System Service.

Figure 9.4: File Upload Pop Up - The user interface for the File Upload Service.
Figure 9.5: Services Administration - User interface for monitoring documents and services.
### 9.2 User Screens

**Figure 9.6: Spring Batch Administration** - The user interface for the File Upload Service.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Execution Count</th>
<th>Launchable</th>
<th>Incrementable</th>
</tr>
</thead>
<tbody>
<tr>
<td>publicationsJobGetHitLead</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobCSTXML</td>
<td>No description</td>
<td>4</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobAGFS</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>well充分Job</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobGrid</td>
<td>No description</td>
<td>3</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>filesReportJobTableToPlotJob</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobOSGI</td>
<td>No description</td>
<td>5</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>fileReportJob</td>
<td>No description</td>
<td>22</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>importHealthNowJob</td>
<td>No description</td>
<td>49</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobStandard</td>
<td>No description</td>
<td>2</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobSCG</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>ifiReportDepthJob</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobRUSG</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobRUD</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobOSIRG</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobOSIRPDF</td>
<td>No description</td>
<td>12</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>publicationsJobRUSD</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>importScoutTicketJob</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>importSpringTechJob</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>tempINvoicesReportJob</td>
<td>No description</td>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>fileUploadJob</td>
<td>No description</td>
<td>14</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Figure 9.7: Spring Batch Administration for Jobs - The user interface for the File Upload Service.
4. ETL Administration  The figure 9.6 shows the name of the jobs registered in the system. On clicking any of the jobs, the detailed executions of the jobs are show as in figure 9.7.

5. CWS  The figure 9.8 shows the user interface exposed by Geoportal. The right of the central panel shows the Catalog and the left panel allows a user to search within the Catalog.

6. WFS  The figure 9.9 shows the user interface exposed by GeoServer and developed using Open Layers. The locations on the map fetch the data associated with their respective points. This data when clicked upon shows the relevant data available for that location.

Figure 9.8: Catalog Service  - The user interface for the Catalog Service exposed by Geoportal.
9. APPENDIX

Figure 9.9: WFS Service - The user interface for the WFS service exposed by GeoServer.

7. Search  The fig 9.10 shows the user interface for Search. This search interface does a full text search on the documents indexed in Solr and then collaborates those results to index into the database to fetch the relevant documents.

Figure 9.10: Search Service - The user interface for the Search service exposed by Solr.

8. Work List  The fig 9.11 shows the conceptual worklist screen. An item would appear on a user’s worklist tracked to the document it is associated with using the UUID. The user can then change the state of the item or if has sufficient permission assign to another person.
9.2 User Screens

Figure 9.11: Worklist Concept Screen - The user interface for the Worklist concept Service.
List of Figures

2.1 Message ................................................................. 7
2.2 Message Channel ..................................................... 7
2.3 Message Endpoint ................................................... 8
2.4 Transformer ............................................................ 8
2.5 Filter ................................................................. 9
2.6 Header Enricher ....................................................... 9
2.7 Router ............................................................... 9
2.8 Splitter ............................................................. 10
2.9 Aggregator .......................................................... 10
2.10 Service Activator .................................................. 11
2.11 Channel Adapter .................................................. 11

3.1 Context Diagram .................................................... 14
3.2 Use Case .......................................................... 16

4.1 SOA ............................................................... 19

5.1 SOA Application Design .............................................. 29
5.2 A generic Module .................................................. 33
5.3 File System Schema ................................................ 41
5.4 Integration Schema ................................................. 41
5.5 Core Schema ....................................................... 42
5.6 Staging Schema .................................................... 43
5.7 ETL Schema ....................................................... 44
5.8 Geoportal Schema ................................................ 45
5.9 Security Schema ................................................... 46
List of Tables

7.1 Coupling Levels ................................................................. 71
7.2 Pair Wise Coupling levels from Fenton and Melton Definition table 7.1 . 72
7.3 Top : Pair Wise Coupling levels from Fenton and Melton Definition table
    Middle : Pairwise coupling values using formula 7.1
    Bottom : Calculation results for coupling of the system ............. 72
7.4 Top : Total Classes Written
    Bottom : Code Written vs Reusing Libraries ......................... 76
References

REFERENCES

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other foreign examination board.

The thesis work was conducted from 07/03/2011 to 08/31/2011 under the supervision of Fabian Moerchen at Siemens Corporate Research, Princeton, NJ, USA.

Barcelona,