Example

XML Legal Document Utility
Software Design Document

Version <1.0>

Rex McElrath

2007-04-20

Revision History

Date	Version	Description	Author
04/18/07	<1.0>	Initial Version of Document	Rex McElrath

Table of Contents

Software Design Document

1 Introduction

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including use case models, sequence diagrams, collaboration models, object behavior models, and other supporting requirement information.

1.1 Purpose

The purpose of the Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to built. The Software Design Document provides information necessary to provide description of the details for the software and system to be built.

1.2 Scope

This Software Design Document is for a base level system which will work as a proof of concept for the use of building a system the provides a base level of functionality to show feasibility for large scale production use. This Software Design is focused on the base level system and critical parts of the system. For this particular Software Design Document, the focus is placed on generation of the documents and modification of the documents. The system will be used in conjunction with other pre-existing systems and will consist largely of a document interaction facade that abstracts document interactions and handling of the document objects.

1.3 Definitions, Acronyms, and Abbreviations

- **Data Objects –** Data objects are Java objects with predefined structures capable of holding data in a structure that is quickly and easily accessible by other parts of the software system. They provide also can help provide a convenient abstraction of the data in a database so that it can be retrieved into a format, such as a denormalized format, that makes access and manipulation of the data easier than if the database had to be called directly. http://java.sun.com/products/jdo/
- **Denormalized** - Normalization of a database is the activity of restructuring the database to avoid data anomalies and inconsistencies by focusing on functional dependencies to help structure the data. A web address to reference about normalization is: http://en.wikipedia.org/wiki/Database_normalization . Denormalization is the act of undoing some of the structural changes made during normalization to help with performance. http://en.wikipedia.org/wiki/Denormalization
- **Digital Signature** – A digital signature is a unique object which is strongly tied to a single entity and the document which signature is intended for. In the same way that a ink on paper signature has characteristics that are unique to a person due to variations in writing a digital signature has characteristics that uniquely tie it to a single person and signing instance. **http://en.wikipedia.org/wiki/Digital_signature**
- **Document Interaction Class, XMLDocumentInteractionEngine** – These are the two terms that will be used to refer to the main software class described within this document.
- **Editable Form Layout**- A user interface presentation layout in which the contents of a document are presented to a user in the format of a form predefined editable areas based on the type of document which is being edited. This type of layout allows for changes to be made in a specific manner so that the data used in the form can be reassembled into a structured data format for transfer to other systems and archival.
- **FOP Libraries** – FOP stands for Formatting Objects Processor. The FOP Processor use an XSL-FO stylesheet and an XML instance to create PDF's, RTF's, and HTML files. FOP libraries bring the functionality of an FOP processor to a library form which can be used within another software program. http://xmlgraphics.apache.org/fop/
- **JDBC/ODBC –** These two acronyms stand for Java Database Connectivity and Open Database Connectivity API's which allow for standardized database access and interaction from software products. JDBC: **http://www.learnthat.com/define/view.asp?id=106** . ODBC: **http://en.wikipedia.org/wiki/ODBC**
- **LegalXML** – A standards body dedicated to issues related to the use of XML in the legal domain, http://www.legalxml.com/
- **PDF** – Portable Document Format, http://en.wikipedia.org/wiki/Portable_Document_Format
- **Pro se** – This is a Latin term which directly translated means “for self” and is used to indicate that a party to a case has chosen to represent them selves to the court instead of choosing for an attorney to represent them to the court. http://en.wikipedia.org/wiki/Pro_se
- **Required Field –** A critical field is a field in a data set for a document that is required for successful document generation. For example, missing parties in a case, missing county location of court, or other data elements that are required to create a valid legal document.

- **Structured Data Format** – A structured data format is data assembled into a discernible structure, such as when data is placed into an XML instance which is validated through the use of an XML schema which defines the structure of the XML document.
- **UUID** – Universally Unique Identifier. A UUID is an identifier standard in software construction which allows for generating identifiers which do not overlap or conflict with other identifiers which were previously created even without knowledge of the other identifiers. **http://en.wikipedia.org/wiki/UUID**
- **Workflow** – The movement of documents through a work process that is structured into tasks with designated persons or systems to perform them and the definition of the order or pathway from start to finish for the work process. http://en.wikipedia.org/wiki/Workflow
- **XML** – eXtensible Markup Language, http://en.wikipedia.org/wiki/XML
- **XSL** – XML Stylesheet Language, which is used to transform and specify formatting for presentations of XML instances. XSL is a family of specifications that include XSLT, XSL-FO, and XPath. XSLT stands for XSL Transform, which is used to transform an XML instance from one form to another. XSL-FO stands for XSL Formatting Objects, which is a specification for formatting objects which format the output of presentations of XML instances in forms such as RTF type files, PDF type files, or HTML files. XPath stands for XML Path Language and is a specification for accessing parts of an XML document using the path to the part in the hierarchy of the XML instance. http://www.w3.org/Style/XSL/

1.4 References

- XML Legal Documents Utility Software Development Plan
  - Version 1.0, Last Updated on 2007-01-31

1.5 Overview

The Software Design Document is divided into 11 sections with various subsections. The sections of the Software Design Document are:

1 Introduction
2 Glossary
3 Use Cases
4 Design Overview
5 System Object Model
6 Object Descriptions
7 Object Collaborations
8 Data Design
9 Dynamic Model
10 Non-functional Requirements
11 Supplementary Documentation

2 Glossary

2.1 Glossary is unused in current document due to Section 1.3 Definitions, Acronyms, and Abbreviations providing terms and definitions for internal use of the document.

3 Use Cases

Use-Case Model Survey

3.1 Actors

3.1.1 Document Manager

3.1.1.1 Information: The Document Manager is a user who works with legal documents. This is an abstraction of the specific users as they all perform similar actions, but for different reasons. For example, a court clerk and an attorney both sign documents, but an attorney does so to state that they created or agree to the documents and the court clerk does so to state that the document has been received and is now secured with a secure hash to detect modification. The mechanics and the processes used for each are the same to apply their respective digital signatures, but the intent and meaning of each application of a digital signature is different. The specific actors who fall into the broader category of document manager are:

3.1.1.1.1 Judge
3.1.1.1.2 Court Clerk
3.1.1.1.3 Attorney
3.1.1.1.4 Paralegal Professional
3.1.1.1.5 Pro Se Party

3.1.1.2 Additional Information: The Document User is the only user seen in the use cases considered essential to the System Under Design. Of the three essential use cases, Create New Document, Generated Document Modification, and Enter Document Into Workflow, the use cases considered the highest priority, Create New Document and Generated Document Modification, have been focused on. Following diagrams in Section 3.3 contain current and future implemented use cases for illustrative purposes of future directions for the System Under Design.

3.1.2 System Under Design

3.1.2.1 The System Under Design is the XML Legal Document System that is being created. This actor represents the system and the actions that it takes.

3.1.3 Administrative User

3.1.3.1 Information: The Administrative User is a user who administers the system by overseeing accounts creation and administration.

3.1.4 Public User

3.1.4.1 Information: The Public User is a generic user to represent a person who is not an attorney or pro se party who will be creating documents but has a valid reason to view and research a document or set of documents in relationship to one or more cases and has been validated through security measures such as signing up for an account in person at the Court Clerk's Office and providing proof of identity.

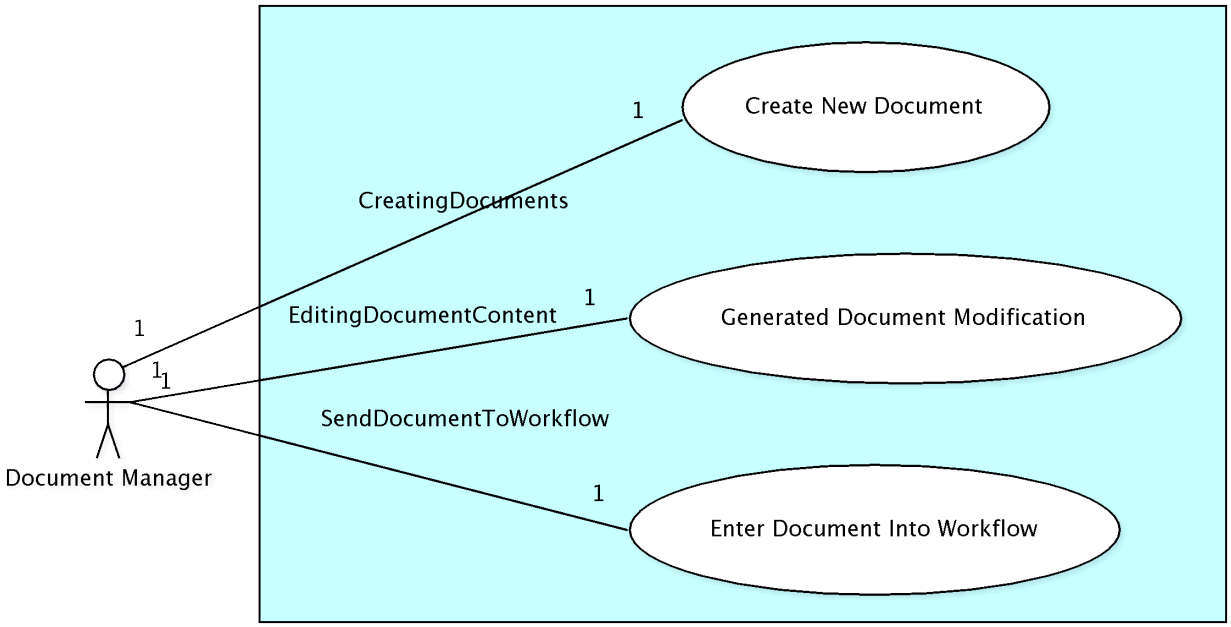
3.2 List of Use Cases

3.2.1 Document Manager User Use Cases

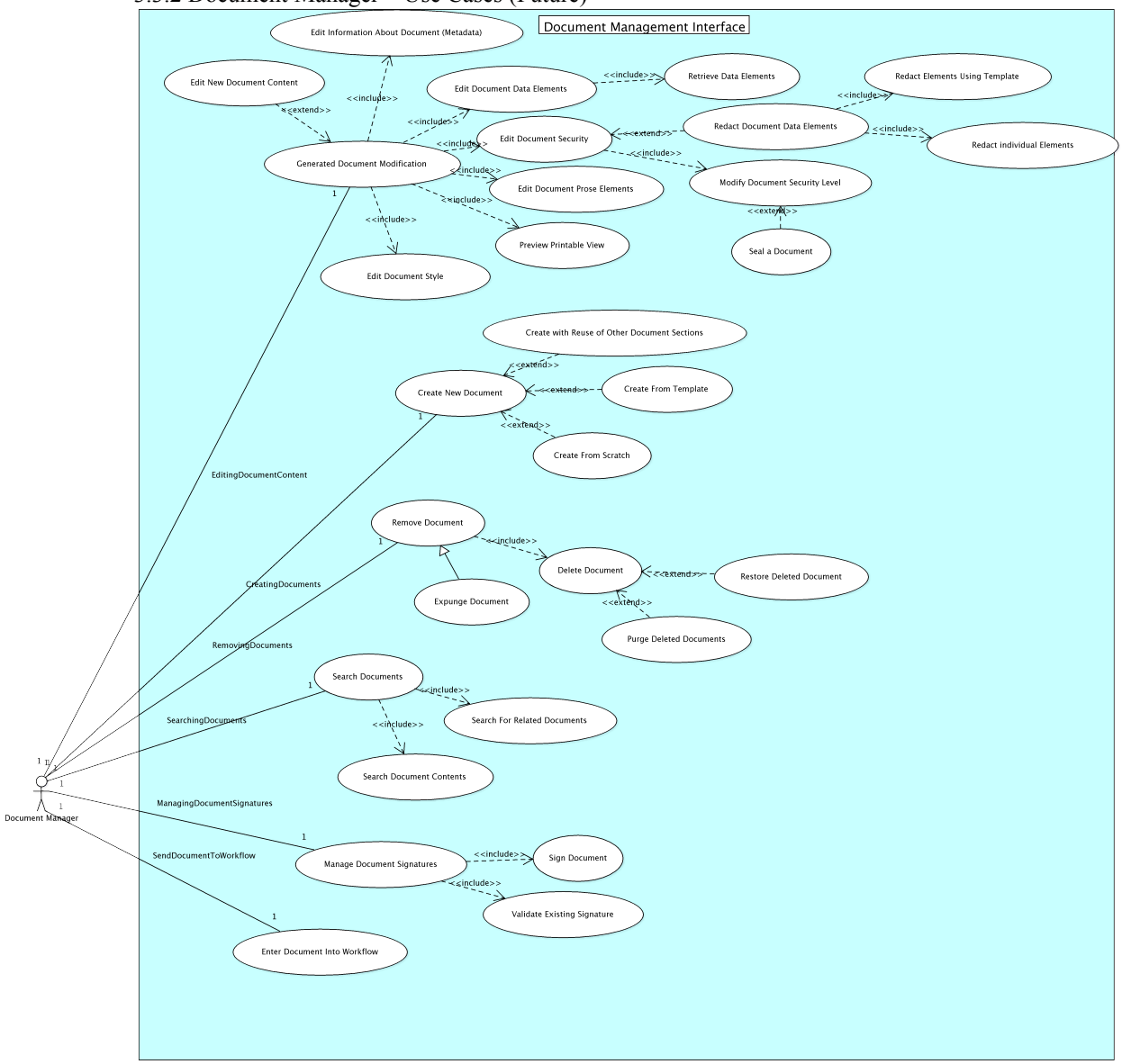
3.2.1.1 Create New Document (Overview)
3.2.1.2 Create New Document(Detail)
3.2.1.3 Generated Document Modification (Overview)
3.2.1.4 Generated Document Modification (Detail)– Element From Data Set
3.2.1.5 Enter Document Into Workflow(Overview)
3.2.1.6 Enter Document Into Workflow(Detail)

3.3 Use Case Diagrams

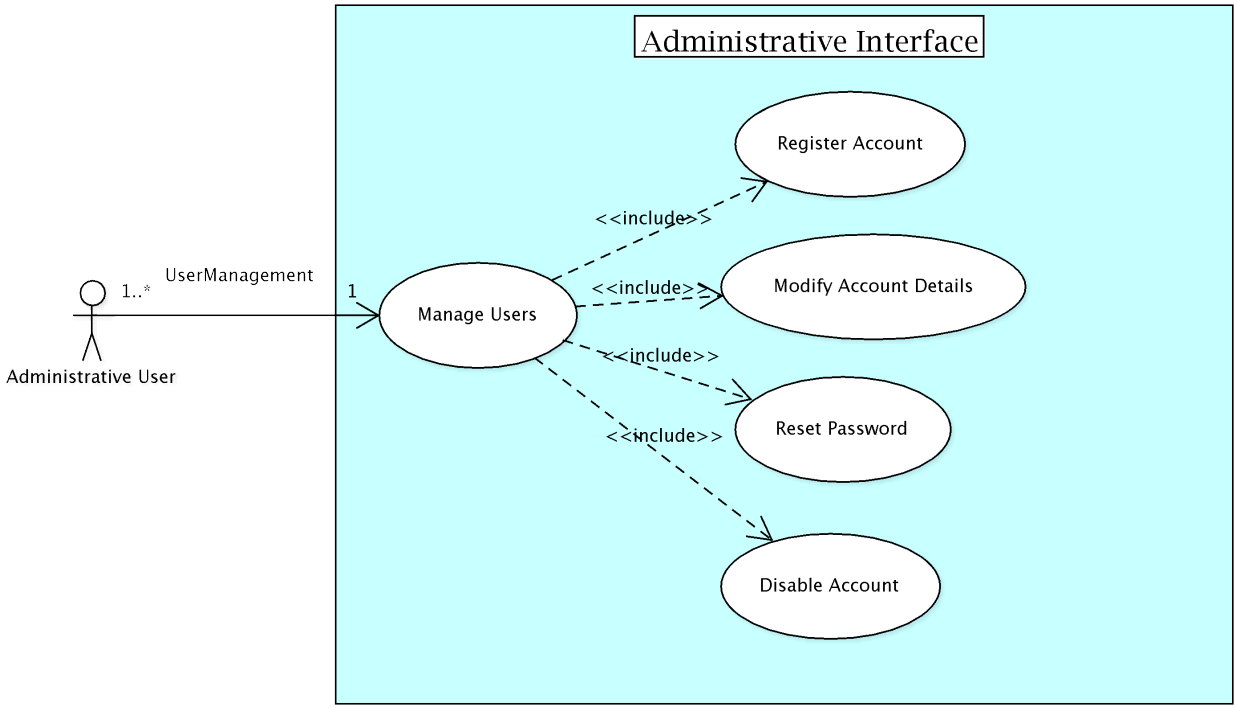
3.3.1 Document Manager- Essential Use Cases ("Enter Document into Workflow" for future update)



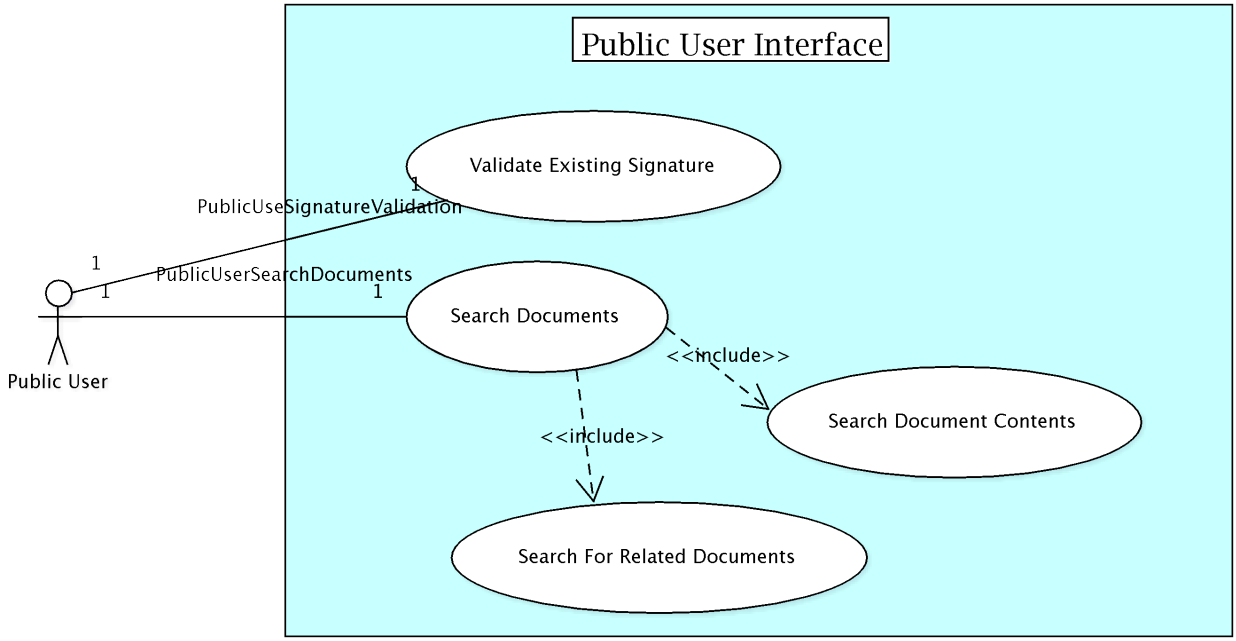
3.3.2 Document Manager – Use Cases (Future)



3.3.3 Administrative User – Use Cases (Future)



3.3.4 Public User – Use Cases (Future)



3.4 Use Cases

3.4.1 Document Manager Use Cases – Create New Document

Use case name: Create New Document		**ID:** *CND*	**Priority:** High
Primary actor: Document Manager	**Source:** Attorneys, Judges	**Use case type:** *Business*	**Level:** Overview
Interested Stakeholders: Judge, Court Clerk, Attorney, Paralegal Professional			
Brief description: This use case describes the creation of a document which is a key function of the system. In this use case, the actor's goal is to generate a document.			
Goal: ● The successful completion of document generation.			
Success Measurement: ● The document is generated and reviewed by the user as acceptable for use.			
Precondition: ● Document Management User has successfully passed through Authentication and Authorization ● Data sufficient to populate all required fields in a data set for a document has been entered into the system that will be used to draw data from to generate the document's data set. **Trigger:** ● Document Management User has reached a point in their workflow in which a document is to be generated.			
Relationships: **Include:** **Extend:** **Depends on:**			
Typical flow of events: 1. A document or set or related documents are selected to be generated. 2. The data from the case management system is pulled by the System Under Design based on the template and case record chosen to populate the document or documents data sets. 3. The Document Management User is allowed to preview the documents and summary of data set used to populate document 4. Once satisfied with the document and data, the user saves the document and enters it into a work flow such as sending to reviewer, or sending for signature.			
Assumptions 1. It is assumed that workflows will be carried out internally or with close partnered agencies that can be interacted with n a similar manner as with an internal system. 2. It is assumed that the case management system will hold appropriate data for use to generate documents. 3. It is assumed that a standardized template for a document is desired instead of using a free form document.			
Implementation Constraints and Specifications:			

3.4.2 Document Manager Use Cases: Create Document (Detail)

Use case name: Create New Document (Detail)		**ID:** *CNDD*	**Priority:** High
Primary actor: Document Manager	**Source:** Attorneys, Judges	**Use case type:** *Business*	**Level:** Detail
Interested Stakeholders: Judge, Court Clerk, Attorney, Paralegal Professional			
Brief description: This use case describes the creation of a document which is a key function of the system. In this use case, the actor's goal is to generate a document.			
Goal: ● The successful completion of document generation.			
Success Measurement: ● The document is generated and reviewed by the user as acceptable for use.			
Precondition: ● Document Management User has successfully passed through Authentication and Authorization ● Data sufficient to populate all required fields in a data set for a document has been entered into the system that will be used to draw data from to generate the document's data set. **Trigger:** ● Document Management User has reached a point in their workflow in which a document is to be generated.			
Relationships: **Include:** **Extend:** **Depends on:**			
Typical flow of events: 1. Document sets are selected to be generated by user by selecting the document type from a presented list or list of document packages. 2. The data from the case management system populates the document sets 3. The System Under Design uses the document or set of documents selected to determine the criteria for pulling data from the case management system, populating the XML instance for a data set for the documents and matching the XML data sets with XSL style sheets. 4. The System Under Design uses predefined security classifications of data elements to include security criteria for elements within XML data sets. 5. Exception: If insufficient data is available to completely populate a document, a notice is given to the user with a summary of missing or incomplete items and the choice to return to the case management system to fill out the missing information or to proceed with document generation if the missing fields are not classified as required fields for the document. 6. Invalid data is not expected as the case management system is expected to handle validation of data before it reaches the point of generating documents. 7. “Populating the document” means populating an XML instance per document that is paired with a specific style sheet so that when previewed, the data and other prose of the document are presented in a single presentation. 8. The user is allowed to preview the documents and summary of data set used to populate document			

9. To change data, return to case management system and update fields
10. The preview for the user is created through the use of combining the XML instance holding the data and the XSL style sheet for the document through the use of a Formatting Objects Processor to create a PDF.
11. Once satisfied with the document and data, the user saves the document and enters it into a work flow (send to reviewer, send for signature, etc)
12. For the System Under Design to move the XML data instance and XSL style sheet together through a workflow, the XSL used for the transform is referenced from within the XML and a 1..1 association is created within the database between the XML instance and the respective style sheet. Since a single XSL can be used many times to create a document, the XSL style sheets are distinctly versioned within the system under design and the specific version used to create the document is noted in the XML and the database association between the XML data set and the XSL style sheet.
13. To route to a new workflow, the document is associated with a new workflow in the database. For example, if a document is to be used for an approval process, then it is referenced by that workflow so that it can be called up by the appropriate person. Specific workflows are out of scope for this system as it is an enabler of workflows, but does not determine how they will be built.

Assumptions

1. It is assumed that workflows will be carried out internally or with close partnered agencies that can be interacted with n a similar manner as with an internal system.
2. It is assumed that the case management system will hold appropriate data for use to generate documents.
3. It is assumed that a standardized template for a document is desired instead of using a free form document.

Implementation Constraints and Specifications:

3.4.3 Document Manager: Generated Document Modification (Overview)

Use case name: Generated Document Modification - Overview		**ID:** *GDMO*	**Priority:** High
Primary actor: Document Manager	**Source:** Attorneys, Judges	**Use case type:** *Business*	**Level:** Overview
Interested Stakeholders: Judge, Court Clerk, Attorney, Paralegal Professional			
Brief description: This use case describes the modification of a data set which modifies the document that is displayed to a user. In this use case, the actor's goal is to modify the data elements of a document.			
Goal: ● The successful completion of document modification.			
Success Measurement: ● The document is modified and reviewed by the user as acceptable for use.			
Precondition: ● Document Management User has successfully passed through Authentication and Authorization ● A document has been generated and is saved for use by a workflow or archive. **Trigger:** ● Document Management User has reached a point in their workflow in which a document is to be modified.			
Relationships: **Include:** **Extend:** **Depends on:**			
Typical flow of events: 1. To modify the data set used for a document a user selects a document to update. 2. To select the document to update, the user uses a text based search or a reference number to locate the document within the System Under Design. 3. Once selected, the user initiates an update data set routine to update the data set which then populates the documents with new data when next previewed.			
Assumptions 1. It is assumed that no structural changes are to be made to standardized template based documents, such as not introducing movement of document sections without creation of new templates that contain the new layout for sections.			
Implementation Constraints and Specifications:			

3.4.4 Document Manager: Generated Document Modification (Detail)– Element From Data Set

Use case name: Generated Document Modification – Element From Data Set	**ID:** *GDMEDS*	**Priority:** High	
Primary actor: Document Manager	**Source:** Attorneys, Judges	**Use case type:** *Business*	**Level:** Detail
Interested Stakeholders: Judge, Court Clerk, Attorney, Paralegal Professional			
Brief description: This use case describes the modification of a data set which modifies the document that is displayed to a user and can initiate an update of the case management system database. The data is new and not already present within the case management system database. In this use case, the actor's goal is to modify the data elements of a document.			
Goal: ● The successful completion of document modification through modifying the elements of the data set of the document.			
Success Measurement: ● The document is modified and reviewed by the user as acceptable for use.			
Precondition: ● Document Management User has successfully passed through Authentication and Authorization ● A document has been generated and is saved for use by a workflow or archive. **Trigger:** ● Document Management User has reached a point in their workflow in which a document is to be modified with data that is not currently in the user database.			
Relationships: **Include:** **Extend:** **Depends on:**			
Typical flow of events: 1. To modify the data set used for a document a user selects a document to update. 2. To select the document to update, the user uses a text based search or a reference number to locate the document within the System Under Design. 3. Once the document is selected, the user selects to edit the documents data elements. 4. The System Under Design reads in the documents data set into memory structures. 5. The System Under Design then presents the user with a screen that has the data from the elements in the data set for the document displayed in manner that allows for them to be reviewed and selected for editing. 6. The System Under Design presents the user with the data elements in an ordered layout with edit options next to each data item, or section of data items. 7. To edit an item, the user clicks on the “Edit” button next to the element, or element set, desired to be edited. 8. When the element, or group of related elements, to edit is selected by the user, the System Under Design takes the user to a data entry page built specifically for that type of data (searches for persons, validations for telephone numbers, etc) and allowed to edit the data. 9. Once edited, the data is validated and reinserted into the data set and data base by the System Under Design using the documents metadata to correctly match the data set with the correct case in the case management system.			

10. Once the XML data set is updated by the System Under Design with the new information, the user is allowed to preview the document to review the updated document.

Assumptions

1. It is assumed that no structural changes are to be made to standardized template based documents, such as not introducing movement of document sections without creation of new templates that contain the new layout for sections.

Implementation Constraints and Specifications:

3.4.5 Document Manager: Enter Document Into Workflow (Overview)

Use case name: Enter Document Into Workflow(Overview)		**ID:** *EDIWO*	**Priority:** Medium
Primary actor: Document Manager	**Source:** Attorneys, Judges	**Use case type:** *Business*	**Level:** Overview
Interested Stakeholders: Judge, Court Clerk, Attorney, Paralegal Professional			
Brief description: This use case describes the entering of a created document into a workflow, such as adding to a package of documents being prepared, sending for review, sending to the court (aka Filing into Court), or sending to case participants. The use case routes the document to the next "inbox" of the workflow by saving the document, updating status of the document, and notifying the target of the workflow that the document is ready to be processed.			
Goal: ● The successful completion of readying a document to be processed as part of a workflow and notification of the intended target that the document is ready to be processed.			
Success Measurement: ● The document is saved with a status of ready to be processed, and the appropriate target has been notified of the status of the document.			
Precondition: ● Document Management User has successfully passed through Authentication and Authorization ● A document has been generated, modifications are completed, and it is ready to be saved for use by a workflow. **Trigger:** ● Document Management User has reached a point in their workflow in which a document is ready to be entered into another workflow.			
Relationships: **Include:** **Extend:** **Depends on:** ● **Create New Document** ● **Generated Document Modification**			
Typical flow of events: 1. Document Manager User has a document that is ready to be entered into a workflow. 2. The System Under Design presents the user with a selection of workflow types. 3. The user selects a type of workflow to use. 4. The System Under Design presents the user with addressing options. 5. The user selects a destination address(es) for the document. 6. The user selects submit to enter the document into the workflow.			
Assumptions 1. The types of workflow are known and there are existing code types and addressing information for notifications to be received.			
Implementation Constraints and Specifications:			

3.4.6 Document Manager: Enter Document Into Workflow (Detail)

Use case name: Enter Document Into Workflow(Detail)		**ID:** *EDIWD*	**Priority:** Medium
Primary actor: Document Manager	**Source:** Attorneys, Judges	**Use case type:** *Business*	**Level:** Detail
Interested Stakeholders: Judge, Court Clerk, Attorney, Paralegal Professional			
Brief description: This use case describes the entering of a created document into a workflow, such as adding to a package of documents being prepared, sending for review, sending to the court (aka Filing into Court), or sending to case participants. The use case routes the document to the next "inbox" of the workflow by saving the document, updating status of the document, and notifying the target of the workflow that the document is ready to be processed.			
Goal: ● The successful completion of readying a document to be processed as part of a workflow and notification of the intended target that the document is ready to be processed.			
Success Measurement: ● The document is saved with a status of ready to be processed, and the appropriate target has been notified of the status of the document.			
Precondition: ● Document Management User has successfully passed through Authentication and Authorization ● A document has been generated, modifications are completed, and it is ready to be saved for use by a workflow. **Trigger:** ● Document Management User has reached a point in their workflow in which a document is ready to be entered into another workflow.			
Relationships: **Include:** **Extend:** **Depends on:** ● **Create New Document** ● **Generated Document Modification**			
Typical flow of events: 1. Document Manager User has a document that is ready to be entered into a workflow. 2. The System Under Design presents the user with a selection of workflow types. (a) The List of Workflows known at this point is: i. Send to Court ii. Send to Judge iii. Send Copy as Secondary Service iv. Send to Peer for Review 3. The user selects a type of workflow to use. 4. The System Under Design captures the workflow type code for the selected by the user to use when submitting the choice. 5. The System Under Design presents the user with addressing options. (a) The addressing options are based on the user's profile and which courts, judges, and service recipients, and peers are configured within their profile as allowable addressing options. 6. The user selects a destination address(es) for the document. 7. The System Under Design Records the destination address(es) id's for use when			

submitting the document to the workflow.

8. The user selects submit to enter the document into the workflow.
9. The System Under Design updates the status of the document to reflect that is has been entered into a workflow to disallow additional edits by the user submitting the document and to allow edits and/or reviewing by the intended recipients of the document.
10. The System Under Design issues notifications that are sent out through email to the intended recipients that the document is ready for action on their part.

Assumptions

1. The types of workflow are known and there are existing code types and addressing information for notifications to be received.

Implementation Constraints and Specifications:

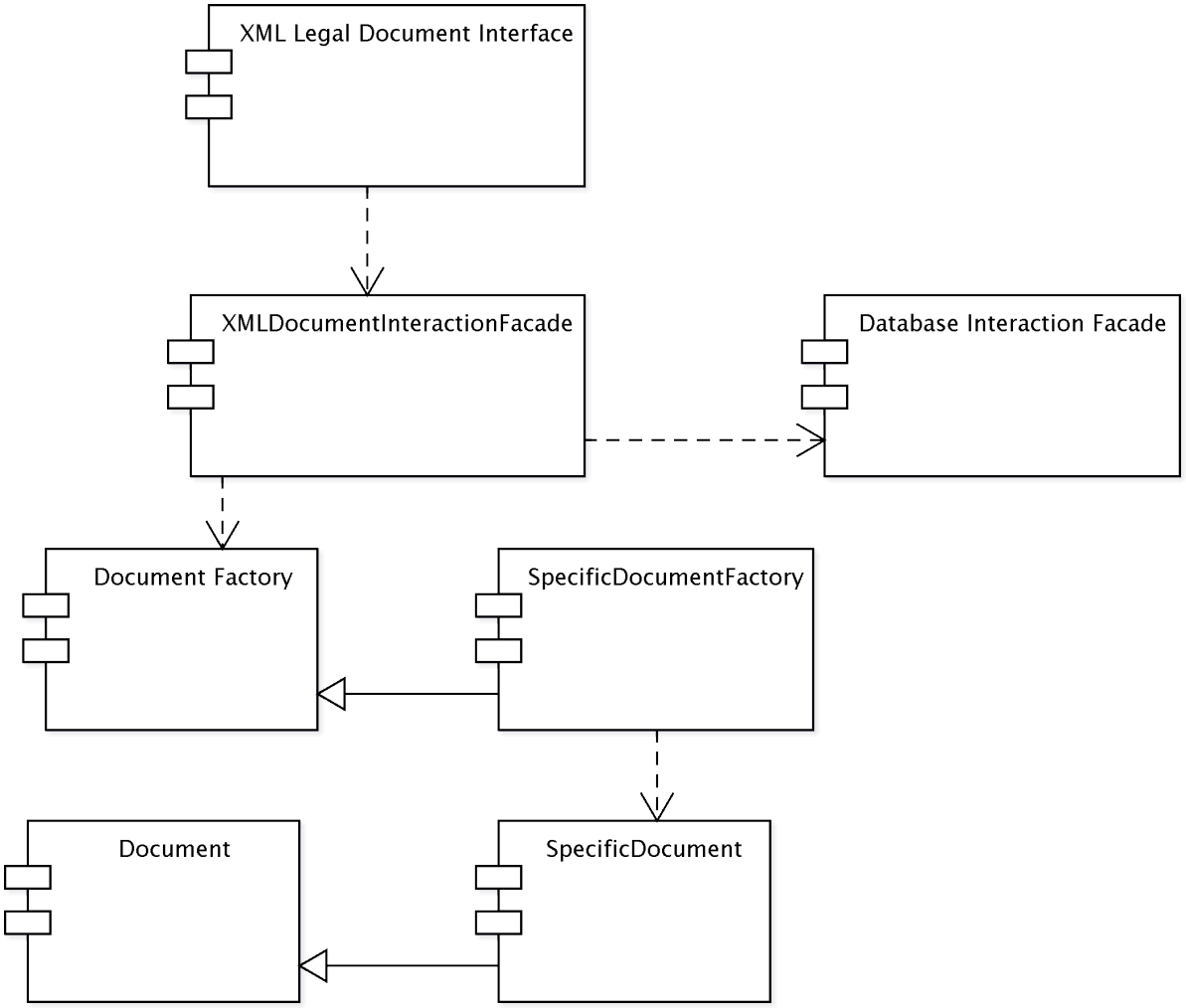
4 Design Overview

4.1 Introduction

The Design Overview is section to introduce and give a brief overview of the design. The System Architecture is a way to give the overall view of a system and to place it into context with external systems. This allows for the reader and user of the document to orient them selves to the design and see a summary before proceeding into the details of the design.

4.2 System Architecture

4.2.1 Overall Structure for the XML Legal Document Utility System



4.3 System Interfaces

4.3.1 External User Interface Requirements

4.3.1.1 User Interfaces

- The user interface for the system will allow the user to easily generated documents, search for documents, and modify documents. The user should be presented with all main functions on the first user interface page to allow for the user to select the function to use without the need to navigate inward to find it. The interface will need to use tab focus marks to allow for navigation using a keyboard as much as possible to alleviate stress on users' arms and hands caused by changing constantly from keyboard to mouse. It will be accessible through a web interface to allow for centralized hosting and use by various operating systems.

4.3.1.2 Software Interfaces

- The software will need to interface with a case management system to pull data from it and push data updates to it. The connection will be a standard database connection using JDBC or ODBC.

4.3.1.3 Communication Interfaces

- The software will need to interface with a case management system to pull data from it and push data updates to it. The connection will be a standard database connection using JDBC or ODBC.

4.4 Constraints and Assumptions

4.4.1 List of Assumptions

4.4.1.1 It is assumed the certain documents used within a court and with closely partnered agencies can be standardized and held stable enough in structure that the supporting structures of an XML schema for the XML data set, an XSL Stylesheet, a classification of the data elements used for the document for security applications, and element update screens can be created and held reasonably stable to avoid a churn of constant modifications to the system and the supporting elements for the documents.

4.4.2 List of Dependencies

4.4.2.1 The system will be dependent on at least one case management system to be able to pull data from. If the case management system is not acceptable to pull data from, such as missing fields required for document generation or unable to allow an adapter or service to be created that allows for the pulling of data to create the documents.

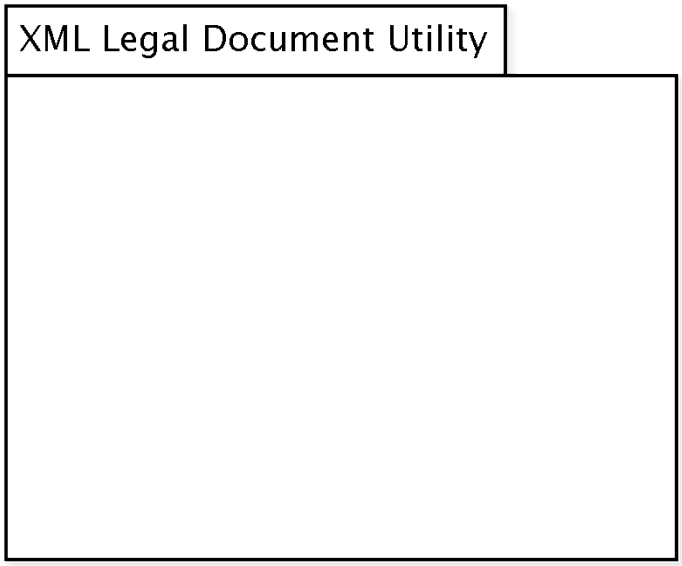
5 System Object Model

5.1 Introduction

The System Object Model Section allows for a description of the subsystems in use. This allows for describing the system in a overall manner to show the different groupings of parts into respective systems. For the System Under Design, only one system is used and no subsystems are specified.

5.2 Subsystems

5.2.1 XML Legal Document Utility Package



5.3 Subsystem Interfaces

5.3.1 None Defined: As the system is contained with in a single package, external interfaces are used but internal interfaces are not necessary.

6 Object Descriptions

6.1 Objects

6.1.1 XMLDocumentInteractionFacade

Class name: XMLDocumentInteractionFacade	
Brief description: The XMLDocumentInteractionFacade class is responsible for controlling actions relating to managing documents. For example, the XMLDocumentManagementFacade class would handle such tasks as searching for and retrieving documents. It is the controller class that abstracts more specific helper classes.	
Attributes (fields)	**Attribute Description**
DocumentFactory docFactory;	This is a declaration of a Document Factory object to be used throughout the class for different methods.
	Program Description Language
	private DocumentFactory docFactory;
AuditLog auditLog;	**Attribute Description**
	This is a declaration of a Audit Log object to be used throughout the class for different methods.
	Program Description Language
	private AuditLog auditLog;
DocumentSigner docSigner;	**Attribute Description**
	This is a declaration of a Document Signer Object to be used throughout the class for different methods.
	Program Description Language
	private DocumentSigner docSigner;
WorkFlow workflow;	**Attribute Description**
	This is a declaration of a WorkFlow object to be used throughout the class for different methods.
	Program Description Language
	private WorkFlow workflow;
Document doc;	**Attribute Description**
	This is a declaration of a Document Object to be used throughout the class for different methods.
	Program Description Language
	private Document doc;
Methods (operations)	**Method Description**

Class name: XMLDocumentInteractionFacade	
searchForDocument()	A method to perform a search for documents, for example by reference number or by text key words, and returns a result set of possible matches.
	Program Description Language

	public Set searchForDocument(String searchString, int searchTypeCode) {
	// Determine search type
	// Types:
	// * Case Number (External or Internal) - 1
	// * Full Text - 2
	if (searchTypeCode == 1) {
	// For now, these are assume to be provided by
	// a facade such as those built with Hibernate
	dbFacade.doCaseNumberSearch(searchString);
	} else if (searchTypeCode == 2) {
	dbFacade.doFullTextSearch(searchString);
	}
	
	return ResultSet;
	}

pullDocument()	**Method Description**
	A method to locate the document and retrieve document structure with contains both the XML data set and XSL stylesheet.
	Program Description Language

	public void pullDocument(Uuid docUuid) {
	
	doc = docFactory.pullDocument(docUuid);
	return null;
	}

checkDocumentCriteria()	**Method Description**
	A method to lookup and retrieve the data criteria for a document.
	Program Description Language

	public String checkDocumentCriteria() {
	String validationResultsHolder;
	
	validationResultsHolder = doc.validateXML();
	
	return validationResultsHolder;
	}

pullDataFromCaseManagement()	**Method Description**
	A method to retrieve data from the case management system based on the data criteria for the document selected and load or refresh the document with the data retrieved.
	Program Description Language

	public void pullDataFromCaseManagement(String caseRefId) {
	// Assumes Case UUID has been searched for and found
	doc.pullData(caseUuid);
	return null;
	}

Class name: XMLDocumentInteractionFacade	
pushDataToCaseManagement()	**Method Description**
	A method to update the case management system when data elements are updated within a document in a workflow.
	Program Description Language

	public void pushDataToCaseManagement() {
	doc.pushData();
	}

setType()	**Method Description**
	A method to set the type of document to generate.
	Program Description Language

	public void setType(int docTypeCode) {
	docFactory.setTypeCode(docTypeCode);
	}

producePDF()	**Method Description**
	A method to combine the XML instance and XSL Stylesheet with the use of FOP libraries to create a PDF to produce to the user of the system.
	Program Description Language

	public URL producePDF(Uuid docUuid) {
	URL pdfLocation;
	pdfLocation = doc.ToPDF(docUuid);
	return pdfLocation;
	}

saveState()	**Method Description**
	A method to save the state of the document.
	Program Description Language

	public String saveState(String docUuid) {
	String persistResult;
	persistResult = doc.persist();
	return persistResult;
	}

setDataElement()	**Method Description**
	A method set which is overloaded and allows for modification of the different types of elements seen in an XML data set for a document.
	Program Description Language

	public void setDataElement(String newData, int elementId) {
	Uuid docUuid = doc.getUuid();
	doc.setDataElement(docUuid, elementId, newData);
	}

associateDocWithWorkFlow()	**Method Description**
	A method set which is used to create an association with a workflow based on the workflow name and parameters passed to it.
	Program Description Language

	public void associateDocWithWorkFlow(String workFlowRefId, Uuid docUuid, Uuid addresses[]) {
	workflow.initiateNewWorkFlow(workFlowRefId, docUuid,
	addresses);
	}

Class name: XMLDocumentInteractionFacade	
signDocument()	**Method Description**
	A method to generate a digital signature and insert it into the XML data set to become part of the document.
	Program Description Language

	public Uuid signDocument(Uuid docUuid, Uuid personUuid) {
	Uuid signatureUuid = signDoc(docUuid, personUuid,);
	return signatureUuid;
	
	}

updateAuditLog()	**Method Description**
	A method to update audit logs based on the actions taken.
	Program Description Language

	public int updateAuditLog(actorUuid, docUuid, actionCode) {
	int auditUpdateSuccessSignal;
	// 0 if successful, 1 if error
	auditLogUpdateSuccessSignal = auditLog.newEntry(actorUuid, docUuid, actionCode);
	return auditLogUpdateSuccessSignal;
	}

readAuditLog()	**Method Description**
	A method to allow for returning a result set containing the audit log for a particular document.
	Program Description Language

	public ResultSet readAuditLog(Uuid docUuid) {
	auditLog.getEntries(docUuid);
	}

generateDocument()	**Method Description**
	A method to generate a document using the type of document to generate and the data source of a case record. The method utilizes a factory pattern class to generate the document.
	Program Description Language

	public String generateDocument(int docTypeCode, Uuid caseUuid) {
	String createdDocUuid;
	docFactory.createDocument(docTypeCode, caseUuid);
	createdDocUuid = doc.getUuid().toString();
	return creatDocUuid.toString;
	}

XMLDocumentInteractionFacade()	**Method Description**
	This is a constructor method for the class.
	Program Description Language

	public XMLDocumentInteractionFacade() {
	docFactory = new DocumentFactory();
	auditLog = new AuditLogFacade();
	docSigner = new DocumentSigner();
	workflow = new WorkFlowFacade();
	}

6.1.2 DataSetEditBackingBean

6.1.2.1 The DataSetEditBackingBean is left without design details temporarily in order to focus on internal mechanics of the system.

Class name: DataSetEditBackingBean	
Brief description: The DataSetEditBackingBean is responsible for supporting the logic and validation of user interface pages used for editing the data held within the XML data set of a document.	
Attributes (fields)	**Attribute Description**
uiInterfaceComponent	This is a generic for attributes used to represent user interface components to be displayed and used in the user interface.
dataComponent	This is a generic for the data holding attributes that will need to be created based on the set of elements to be edited.
Methods (operations)	**Method Description**
getUIInterfaceComponent()	This is a generic for the getters used for user interface backing beans.
setUIInterfaceComponent()	This is a generic for the setters used for user interface backing beans.
getDataComponent()	This is a generic for the getters used for data used with the interface for documents.
setDataComponent()	This is a generic for the setters used for data used with the interface for documents.
validate(data, validationType)	An overloaded method set which is used to validate modifications to a data set based on the type of element being edited.
DataSetEditBackingBean()	This is a constructor method for the class.

6.1.3 DocumetFactory

Class name: DocumentFactory	
Brief description: This is a class to create an abstraction of document creation and help broker creation of different document types.	
Attributes (fields)	**Attribute Description**
Document doc[];	This is an object array to hold document objects as they are created.
	Program Description Language
	private Document doc[];
int activeDocuments;	**Attribute Description**
	This is a variable used to keep count of the documents created by each factory instance. This variable is used in methods, but is in place for future use when multiple documents will be able to be used and manipulated at the same time by a user.
	Program Description Language
	private int activeDocuments;
Methods (operations)	
createDocument()	**Method Description**
	This is a method used to abstract the creation of specific document types by being the broker to call the specific document creation factories.
	Program Description Language

	public Document createDocument()(int docTypeCode, Uuid caseUuid)
	{
	
	if (docTypeCode == 1) {
	SpecificDocument1Factory specDoc1Factory =
	new SpecificDocumentFactory();
	doc[activeDocuments] =
	specDoc1Factory.createDocument(caseUuid);
	activeDocuments++;
	} else if (docTypeCode == 2) {
	SpecificDocument2Factory specDoc2Factory =
	new SpecificDocument2Factory();
	doc[activeDocuments] =
	specDoc2Factory.createDocument(caseUuid);
	activeDocuments++;
	} // . . . for all doc types
	
	return doc[activeDocuments -1];
	}

Class name: DocumentFactory	
pullDocument()	**Method Description**
	This is a method to generate a document object
	Program Description Language

	public Document pullDocument(UUID docUuid) {
	dbFacade = new DatabaseInteractionFacade();
	dbFacade.pullDocumentXMLFromDB(docUuid);
	// DB facade retrieves XML data and style instances
	// XML objects are read into memory to create an in memory
	// structure of the XML to create document
	// Created document reference is returned to the caller.
	return doc;
	}

DocumentFactory()	**Method Description**
	This is a default constructor method.
	Program Description Language
	public void DocumentFactory() { }

6.1.4 SpecificDocumentFactory

Class name: SpecificDocumentFactory	
Brief description:	
Attributes (fields)	**Attribute Description**
DatabaseInteractionFacade dbFacade;	This is a declaration of a Database Interaction Facade object to be used throughout the class.
	Program Description Language
	private DatabaseInteractionFacade dbFacade;
Methods (operations)	
createDocument()	**Method Description**
	This is a method to create a document object and to ready that object for use with data and style structures populated and ready for use.
	Program Description Language

```
public Document createDocument(Uuid caseUuid) {

    // 1. Instantiate Document Object (which in turns instantiates an
    //    XML data instance.)
    // 2. Create connection with Database (handled internally by db
    //     facade)
    // 3. Pull Case Related Data
    // 4. Fill Out XML data Instance for this type of document from
    //     the case data
    // 5. Pull up Stylesheet reference for this type of document and set
    //     reference to it.
    // 6. Once data instance and style sheet reference are set, pass
    //    back document object.

    // Declare a document instance
    SpecificDocument doc;
    // Declare a XML data instance
    XMLInstanceData data;
    // Declare a XML style instance
    XMLInstanceStyle style;
    // Declare a Case Record instance
    CaseRecord cr;

    // Instantiate the new document
    doc = new SpecificDocument();
    // Populate the case record with data
    cr = dbFacade.pullCaseData(caseUuid);
    // Based on what the document type is, transfer data from the case
       record
    // to the XML Data instance to populate it.
    populateXMLData(cr);
    // Document style associated by document type
    // Style Instance instantiated for Document in constructor
    // for document.
    return doc;
}
```

Class name: SpecificDocumentFactory	
populateXMLData	**Method Description**
	This is a method to populate the XML data instance part of a document with data from a case record data structure.
	Program Description Language

	private void populateXMLData(CaseRecord cr) {
	// Populate the data set for the document
	doc.setCourt(cr.getCourt());
	doc.setInitiatingPartyAttorney(cr.getInitiatingPartyAttorney());
	doc.setDefendantPartAttorney(cr.getDefendantPartyAttorney());
	...
	// Continue until all needed data elements are populated into the
	// document data structure.
	/* Need to include a document data set for reference */
	}

SpecificDocumentFactory	**Method Description**
	This is a default constructor for the class.
	Program Description Language
	public void SpecificDocumentFactory() { }

6.1.5 Document

Class name: Document	
Brief description:	
Attributes (fields)	**Attribute Description**
General Document Data Elements	This is a grouping of data elements that are common to all documents which will be inherited by subclasses which will create specific documents.
	Program Description Language

	/* Data Elements with cardinality in XML instances */
	/* Elements with cardinality greater than 1:1 are assumed
	* to be arrays of elements
	*/
	/* All are set to be private to the class */
	// Case Caption Information
	Court // 1:1
	Internal Tracking Number // 1:1
	Case Reference Number // 1:1
	External Case Number // 0:*
	Case Style // 1:1
	Alias Case Style // 0:*
	// Parties
	Initiating Party // 1:*
	Responding Party // 1:*
	Initiating Party Attorney // 0:*
	Responding Party Attorny // 0:*
	Witness // 0:*
	Related Party // 0:*
	// Matters
	Matter Code // 1:*
	// Prose Sections
	Prose Elements // 1:*
	// Prose Locations
	Prose Location Info // 1:*

DatabaseInteractionFacade dbFacade	**Attribute Description**
	This is a declaration of a Database Interaction Facade object to be used throughout the class.
	Program Description Language
	// Declare a DB Facade private DatabaseInteractionFacade dbFacade = new DatabaseInteractionFacade();
HashMap hmapElementToId;	**Attribute Description**
	This is a variable to create a hash map for keeping track of identifiers for the data elements for a document. This allows data elements to be referenced by id number instead of by String based name.
	Program Description Language
	// Declare Hashmap for mapping elements to id's private HashMap hmapElementToId;

Class name: Document	
int numberOfDataElements	**Attribute Description**
	This is an variable to help keep track of the number of data elements to be used as in index variable for loops or for working with the hash map.
	Program Description Language

	// Declare an element to be used to hold the number of data elements
	// present.
	private int numberOfDataElements = 1;

int changeTrackerForElements[];	**Attribute Description**
	This is a integer array for use in tracking changes to the data in use so that it can be synchronized with the case management system using the appropriate update methods of the database facade.
	Program Description Language

	// Declare an array to hold id's of elements that have been updated
	// 0's indicate no change, 1's represent change and need to
	// synchronize
	// If there is a 1 present in (elementId - 1) position of the array,
	// then the element has been updated and needs to be
	// synchronized back to the database
	private int changeTrackerForElements[];

XMLInstanceData xmlData;	**Attribute Description**
	This is a declaration of the document object model type of holder for the XML instance responsible for data when the data from the Document object is transferred or read from XML.
	Program Description Language
	// Declare a XML Data Object for data private XMLInstanceData xmlData;
XMLInstanceStyle xmlStyle;	**Attribute Description**
	This is a declaration of the document object model type of holder for the XML instance responsible for stylesheet data when needed for use by the Document object.
	Program Description Language
	// Declare a XSL Object for Style private XMLInstanceStyle xmlStyle;
int docType;	**Attribute Description**
	This is an integer to hold the document type code.
	Program Description Language
	// Holds the Document type code for the document. private int docType;
Methods (operations)	
Data Element Getter	**Method Description**
	This is a place holder for the simple “getter” methods for the general document data elements contained within this class.
	Program Description Language
	public String getValueOfSpecificDataElement() { }

Class name: Document	
Data Element Setter	**Method Description**
	This is a place holder for the simple "setter" methods for the general document data elements contained within this class.
	Program Description Language

	public void setValueOfSpecificDataElement(String value) {
	GeneralDataElement = value;
	}

Document()	**Method Description**
	This is a constructor for the class which allows for the instantiation of the XML data and style structure objects, the setting of the
	Program Description Language

	public void Document(int docTypeCode) {
	docType = docTypeCode;
	// InstantiateDOM object for dealing with data
	xmlData = new XMLInstanceData();
	// InstantiateDOM object for dealing with style data
	xmlStyle = new XMLInstanceStyle();
	// Load style sheet info based on the Document Type
	xmlStyle.loadStyle(docType);
	}

hashElementsToId()	**Method Description**
	HashElementsToID() is a method which adds the data elements to an id structure. The reason for using a hash map is to allow for easy lookup of elements by id number, working with a list of ids or elements, or getting a list of the mapped elements.
	Program Description Language

	// Code to element mapping
	public void hashElementsToId() {
	hmapElementToId = new HashMap();
	hmapElementToId.put(numberOfDataElements++, Court);
	hmapElementToId.put(numberOfDataElements++, Internal Tracking Number);
	hmapElementToId.put(numberOfDataElements++, Case Reference Number);
	hmapElementToId.put(numberOfDataElements++, External Case Number);
	hmapElementToId.put(numberOfDataElements++, Case Style);
	. . .
	// add all elements to the hashmap
	
	// Initialize Change Tracker Array
	changeTrackerForElements[numberOfDataElements];
	}

Class name: Document	
setDataElement()	**Method Description**
	This method allows for an abstraction of setting data elements. It allows for setters to be located by id number for a proxy method for other setter methods.
	Program Description Language

	public void setDataElement(String value, int elementId) {
	hmapToSetterLocater(elementId, value);
	}

getDataElement()	**Method Description**
	This method allows for an abstraction of getting data elements. It allows for getters to be located by id number for a proxy method for other getter methods.
	Program Description Language

	public String getDataElement(int elementId) {
	String value;
	value = hmapToGetterLocator(elementId);
	return value;
	}

hmapToSetterLocator()	**Method Description**
	This is a method to allow for setting data elements based on id. It is a private method used by setDataElement() method.
	Program Description Language

	private void hmapToSetterLocator(int id, String value) {
	if (id == 1) {
	setCourt(value);
	changeTrackerForElements[id] = 1;
	} else if (id == 2) {
	setInternalTrackingNumber(value);
	changeTrackerForElements[id] = 1;
	} else if (id == 3) {
	setCaseReferenceNumber(value);
	changeTrackerForElements[id] = 1;
	} else if . . .
	// add locater statements for all elements
	return void;
	}

Class name: Document

hmapToGetterLocator()

Method Description

This is a method to allow for getting values of elements by using id numbers.

Program Description Language

```
private String hmapToGetterLocator(int id) {
        String value;
        if (id == 1) {
            value = getCourt();
            return value;
        } else if (id == 2) {
            value = getInternalTrackingNumber();
            return value;
        } else if (id == 3) {
            value = getCaseReferenceNumber();
            return value;
        } else if // . . .
        // add locater statements for all elements
    }
```

pushData()

Method Description

This is method for synchronizing the changed data in the document with the case management system database.

Program Description Language

```
public void pushData() {
        int i;
        Uuid caseRecordUuid = getCaseRecordUuid();
        if (changeTrackerForElements[i++ - 1] == 1) {
            dbFacade.updateCaseRecordCourt(caseRecordUuid,
                getCourt());
        } else if (changeTrackerForElements[i++ - 1] == 1) {
            dbFacade.updateCaseRecordInternalTrackingNumber(
                caseRecordUuid, getInternalTrackingNumber());
        } else if (changeTrackerForElements[i++ - 1] == 1) {
            dbFacade.updateCaseRecordCaseReferenceNumber(
                caseRecordUuid, getCourt());
        // . . .
        // Continue until all options for changes have been checked
        // and updated.
    }
```

6.1.6 SpecificDocument

Class name: Specific Document	
Brief description:	
Attributes (fields)	**Attribute Description**
Specific Document Data Elements	This is a grouping of data elements that are common to all documents which will be inherited by subclasses which will create specific documents.
	Program Description Language

	/* Document Specific Data Elements with cardinality in XML */
	/* Elements with cardinality greater than 1:1 are assumed
	* to be arrays of elements
	*/
	/* All are set to be private to the class */
	// Assuming a Summons
	// Appearance Info
	Appearance Date // 1:1
	Appearance Time // 1:1
	Court House Identifier // 1:1
	// Service Info
	Respondent Party Work Address // 1:1
	// More addresses can be added on service info sheet
	. . . .

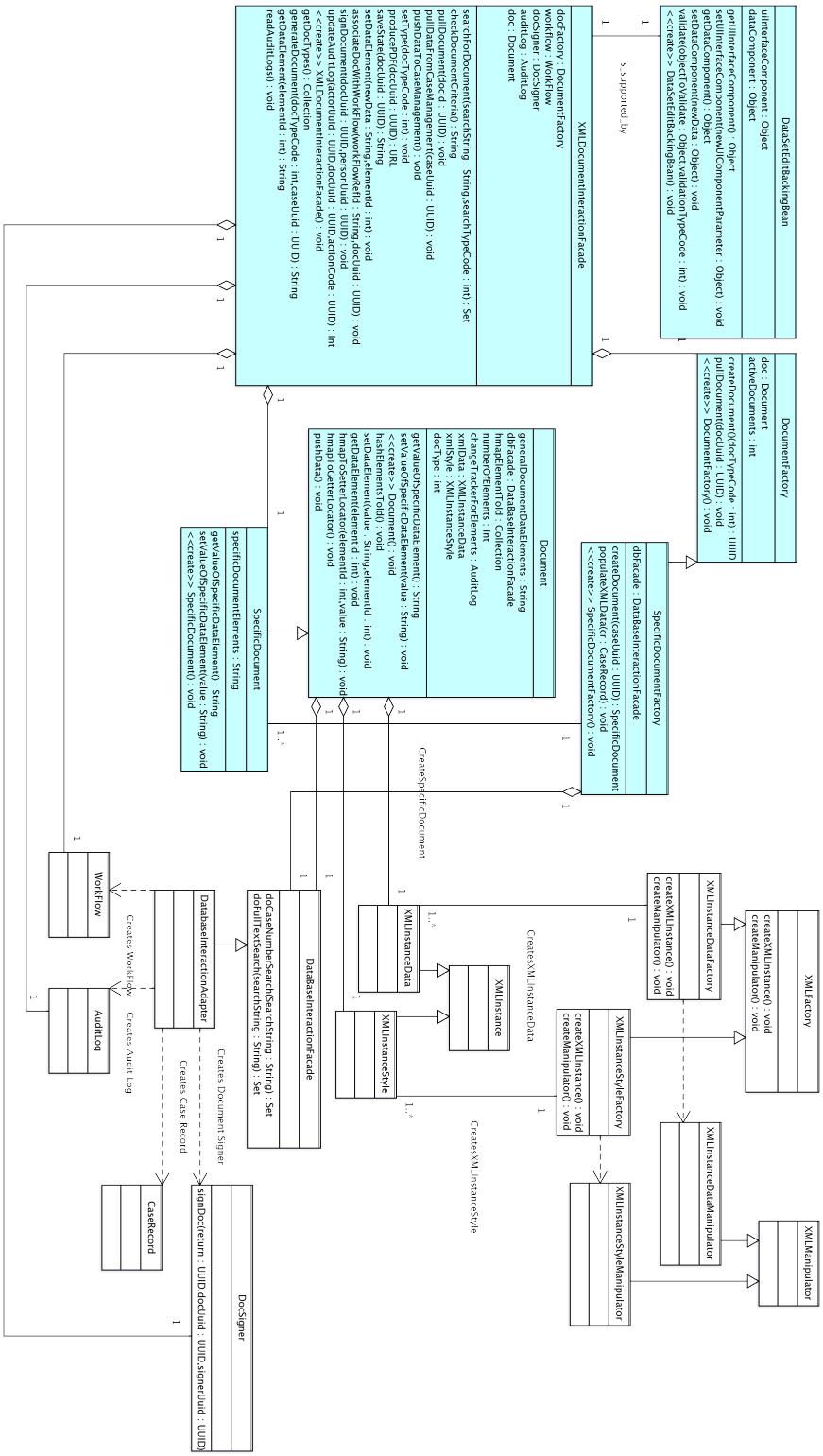
Methods (operations)	
Data Element Getter	**Attribute Description**
	This is a place holder for the simple “getter” methods for the general document data elements contained within this class.
	Program Description Language
	public String getValueOfSpecificDataElement() { }
Data Element Setter	**Attribute Description**
	This is a place holder for the simple “setter” methods for the general document data elements contained within this class.
	Program Description Language

	public void setValueOfSpecificDataElement(String value) {
	GeneralDataElement = value;
	}

7 Object Collaboration

7.1 Object Collaboration Diagram

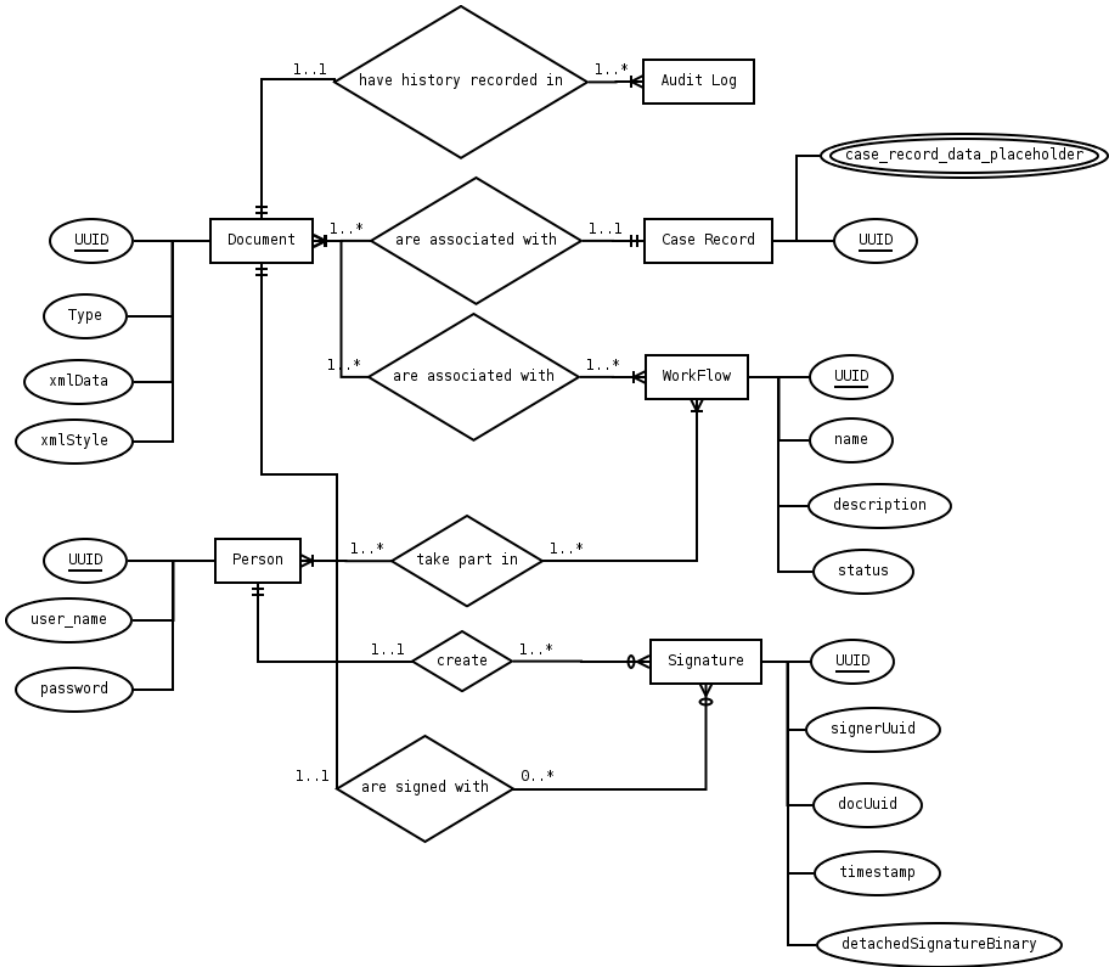
7.1.1 This is a diagram depicting the object relationships. Items in blue are described in Section 6.



8 Data Design

8.1 Entity Relationship Diagram

8.1.1 Basic Entity Relationship Diagram

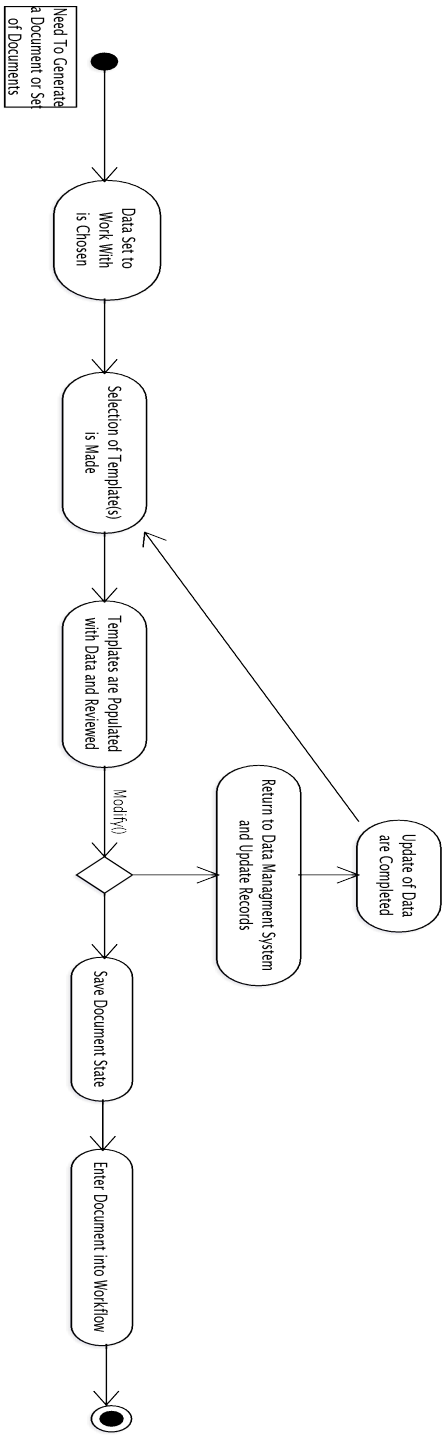


9 Dynamic Model

9.1 Sequence Diagrams

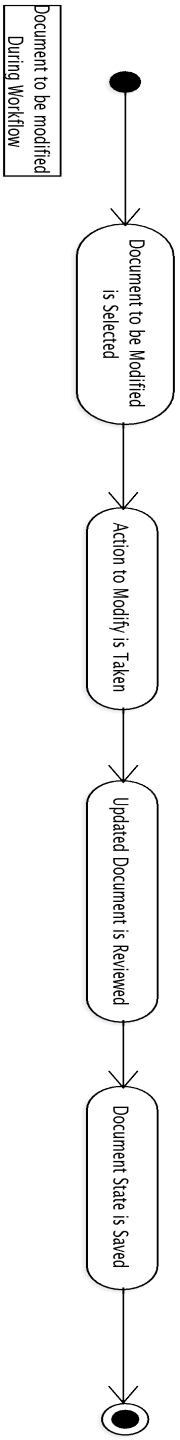
9.1.1 Document Generation Sequence Diagram

9.1.1.1 This diagram show the overview level sequence form moving from data and template to a document.



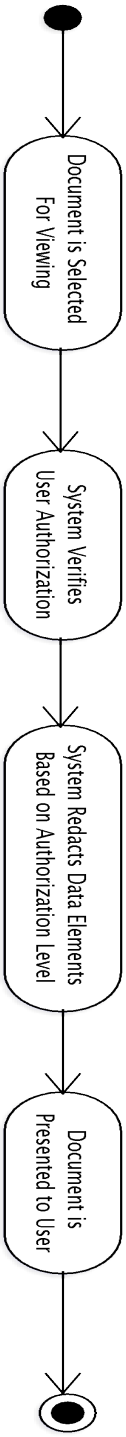
9.1.2 Edit Document Sequence Diagram

9.1.2.1 This diagram shows the overview level sequence for moving from an unmodified original document to a modified document.



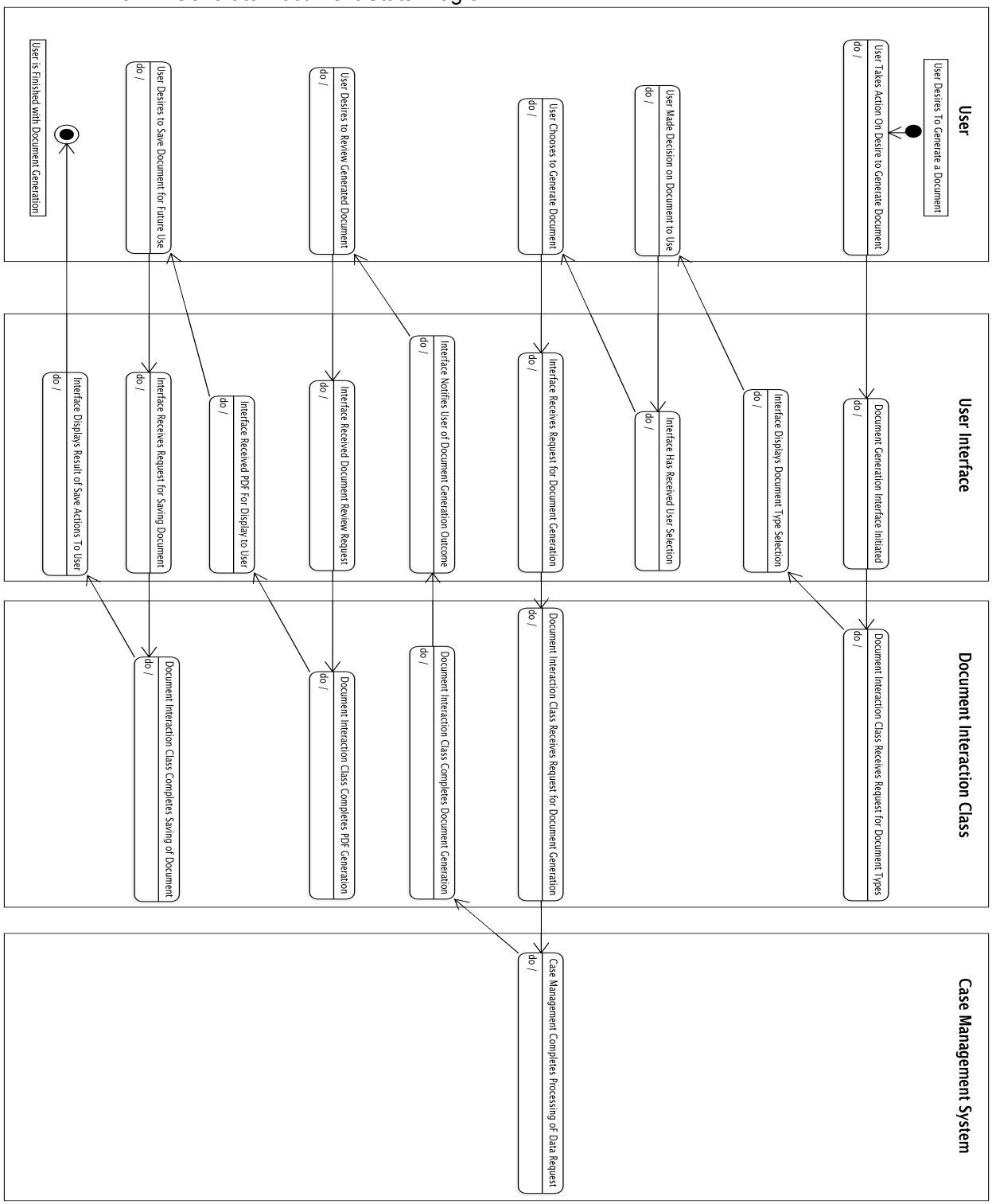
9.1.3 Creation of Human Viewable Presentation

9.1.3.1 This diagram shows the overview level sequence for creating a human presentable view of document from data set and stylesheet to PDF.

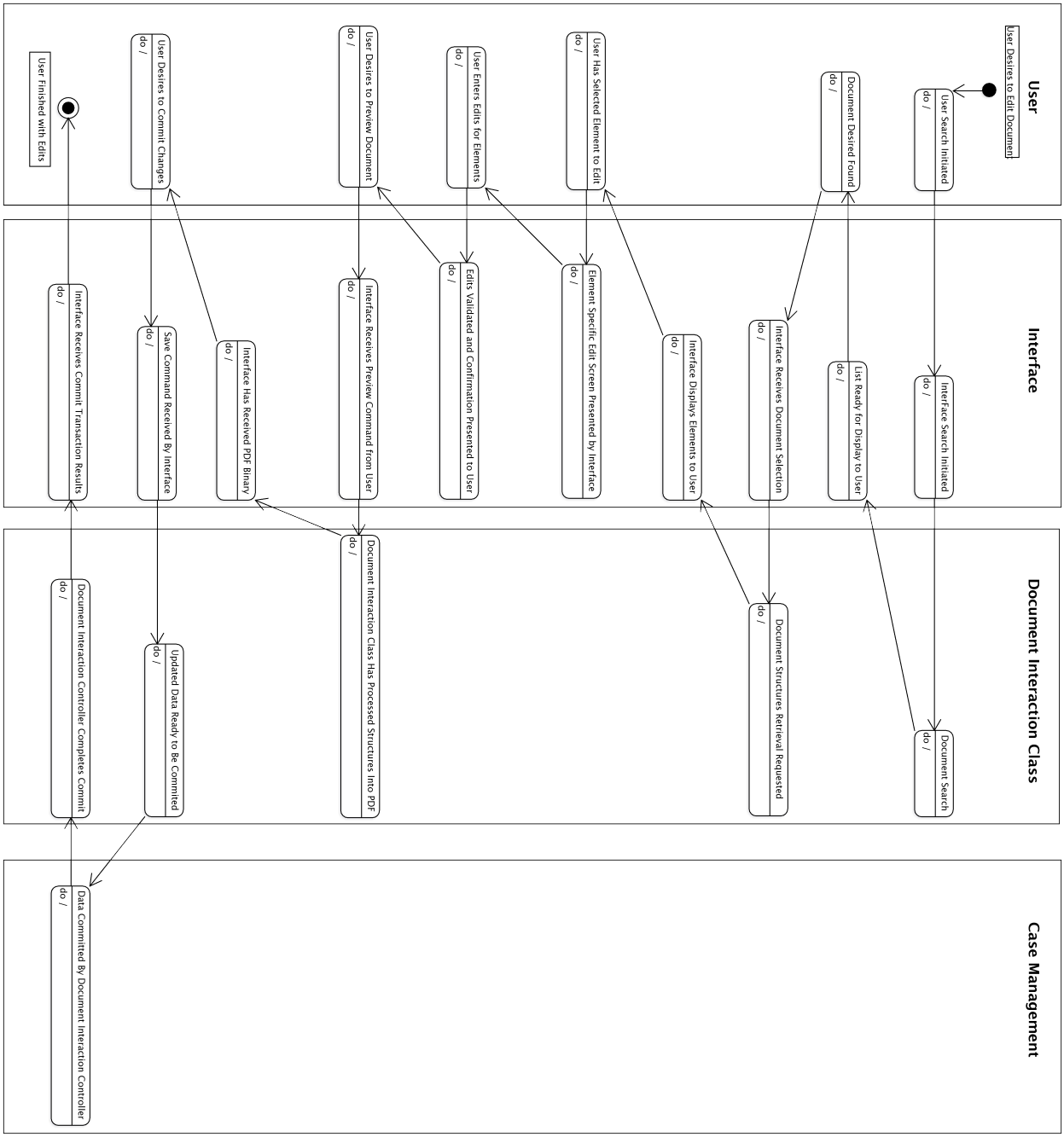


9.2 State Diagrams

9.2.1 Generate Document State Diagram



9.2.2 Edit Document State Diagram



10 Non-functional Requirements

10.1 Performance Requirements

- The system should be able to generate previews of documents within 15 seconds of user request.
- The system should be able to be multi-tasking to allow multiple users, up to 40 simultaneous users per interface instance to interact with the system without having to wait on others to finish working with the system.
- The system should be able to hold and search through large amounts of documents. The data structures used for the system will be fairly simple consisting of a few fields to hold document types and their related codes, XML instances with an id, and an audit log table, however the size of the simple data structures could potentially be quite large.
    - Expected capacity for large volume courts – approximately 108, 000 new documents a year with expected retention capacity of 10 years of active documents. After 10 years, documents can be stored in slower to access storage media. Which equates to approximately 1,080,000 documents that will need to be able to be stored and searched.

10.2 Design Constraints

- The software to be built should take advantage of open source libraries and supporting software, such as databases and web containers, unless an adequate open source product is not available or creatable for use.
    - The work will be licensed under an existing open source license, available at, http://license.gaje.us , and donated for use to standards committees that the agency participates in, such as the LegalXML Technical Committee.
- The software should adhere to locally or nationally recognized standards.
    - XML schemas should follow the National Information Exchange Naming and Design Rules, http://www.niem.gov/topicIndex.php?topic=file-ndr-0_3 .

  When implemented in later versions, document retention schedules should follow the guidelines set forth by the Administrative Office of the Courts in Georgia for records retention, http://www.georgiacourts.org/aoc/records.php .

11 Supplementary Documentation

11.1 Tools Used to Create Diagrams

11.1.1 UML Modeling Tools

11.1.1.1 ArgoUML – Version 0.24, http://argouml.tigris.org/

11.1.2 Entity Relationship Diagramming Tools

11.1.2.1 Dia – Version 0.95, http://live.gnome.org/Dia