DISCUSSION DRAFT
11-16-04

OPEN XML COURT INTERFACE

Electronic Filing Manager Design Document

November 16, 2004

TABLE OF CONTENTS

Page

TABLE OF CONTENTS

(continued)

I. OVERVIEW

I. OVERVIEW

The Open eXtensible Markup Language (XML) Court Interface (OXCI) consortium of state courts intends to produce a middleware implementation for electronic filing for use within all levels of state courts for the receipt, transmission, and validation of electronic filings, court orders, and associated data. The requirements for this middleware are described in the OXCI EFM Software Requirements document.

A. PURPOSE

The purpose of this document is to provide a detailed, technical description of OpenEFM, specifically in regard to OXCI EFM software requirements. Detailed explanations regarding fulfillment of the requirements will be described. This document is intended to be useful for developers and court information technology staff.

B. SCOPE

This document will provide both high-level and detailed descriptions of the design components. The section titled Development Requirements will give a detailed description of how all requirements not yet meet by OpenEFM will be implemented.

C. RELATED DOCUMENTS

References to the following documents may be found within this design document:

- OXCI EFM Architecture.
- OXCI EFM Software Requirements.
- OXCI EFM Court Filing XML Schema.

D. DOCUMENT ORGANIZATION

Section II presents a high-level design. The architectural approach to OpenEFM is described, and several key components are discussed.

Section III presents a detailed design. This section provides explicit, technical descriptions of each module used by OpenEFM. The detailed design section will describe how OpenEFM is implemented currently.

Section IV focuses on development requirements. This section will outline the implementation process of requirements defined in the OXCI EFM Software Requirements document that have been satisfied by OpenEFM. The development requirements will be presented in a detailed, technical manner.

Section V describes the graphical user interface.

APPENDIX A includes a list of definitions and acronyms used throughout the document.

APPENDIX B provides a list of methods defined by the InternalLxmlFiling interface.

APPENDIX C provides class diagrams for the key modules described in this document.

APPENDIX D contains a revision history of this document.

II. HIGH-LEVEL DESIGN

II. HIGH-LEVEL DESIGN

OpenEFM was designed in a modular fashion. Interfaces are provided for key system modules. OpenEFM provides various implementations of these individual module interfaces. A controller object manages the module instances and their configuration, as well as the starting and stopping of the server itself. This object drives the EFM and offers a glimpse into a general overview of the design.

In general, the modules to be used are defined in an XML configuration file. This file contains all of the needed module attributes used for proper configuration. The controller parses the file and instantiates module objects, configuring them as needed.

OpenEFM also contains a basic object model structure. This model structure defines all of the data sets that various modules of the system may need to work with. In general the object models represent groups like EFSPs, users, filings, etc.

Class diagrams for the OpenEFMController are available in APPENDIX C.

III. DETAILED DESIGN

III. DETAILED DESIGN

This section will first examine the controller object. Then the object model hierarchy will be evaluated. All modules and objects discussed in this section have corresponding class diagrams, which are located in APPENDIX C.

A. OPEN EFM CONTROLLER OBJECT

The OpenEFMController object handles the general actions of the server. It reads and parses an XML-based configuration file and controls various tasks associated with administrating the server, such as starting and stopping the EFM. The OpenEFMController object also contains the active instances of the different modules being used by the server. Currently, module interfaces are defined for transceivers, security management, general data model, CMS connector, LegalXML validator, audit logger, ID factory, billing, court policy, virus checking, and the user interface. The controller object provides getters and setters for the current modules being used by the system. An XML-based configuration file specifies which implementations of the module interfaces are to be used. When the OpenEFMServer object reads the configuration file, it creates an OpenEFMController instance. As class implementations of module interfaces are specified in the configuration file, the server objects instantiate the class and set it in the controller object.

A class diagram of the OpenEFMController object is provided in APPENDIX C.

B. DATA OBJECT MODEL

OpenEFM currently provides a set of interfaces defining the internal data objects used. Various system modules use these objects in order to provide the required functionality. For each data object that is modeled in OpenEFM two interfaces are provided. The first interface defines the data set of the object. The second interface manages access to the objects. By defining each object as an interface, modules of the system are able to continue functioning even if a given implementation of an interface is modified to suit a specific court's need. For instance, object managers handle access to all objects, so if a requested object's data is stored in a database, the manager will communicate with the database to retrieve the values. Therefore, if the database requirements change, only the management classes will need to be reimplemented or modified. The benefit of this approach has been demonstrated by the fact that the EFM can currently use several different types of data stores. The EFM can work with a Java 2 Enterprise Edition (J2EE) Enterprise Java Bean(EJB) -based relational database or with the object-persistent data store known as Ozone.

The objects used by OpenEFM are divided into several categories. These categories are user objects, EFSP objects, ID objects, query objects, and LegalXML objects.

1. User Object Model

The User object contains information needed to represent an end user of OpenEFM. This data set includes name, username, password, and role. These values can all be manipulated through methods defined in the object interface. These methods include:

```
public void setName(String name) throws ModelException;
String getName() throws ModelException;
public void setUsername(String username) throws ModelException;
String getUsername() throws ModelException;
public void setPassword(String password) throws ModelException;
String getPassword() throws ModelException;
public void setRole(String roleCode) throws ModelException;
String getRole() throws ModelException;
boolean hasRole(String roleCode) throws ModelException;
```

The UserManager object controls access to the User objects. This includes the ability to create, add, get, and remove individual users. It also allows for search by username for a specific user and getting a list of all users. The methods defined to allow for this are as follows:

```
static final String NAME = "UserManager";
User createUser() throws ModelException;
User getUser(Id userId) throws ModelException;
void removeUser(User user) throws ModelException;
void updateUser(User user) throws ModelException;
User findUserByUsername(String username) throws ModelException;
List getAllUsers() throws ModelException;
```

2. EFSP Object Model

The EFSP object encapsulates the data needed by OpenEFM in regard to the EFSP. This information contains name, username, password, and contact information, along with fields for telephone number and for notes. The interface defines the following methods:

```
public void setName(String name) throws ModelException;
String getName() throws ModelException;
public void setUsername(String username) throws ModelException;
String getUsername() throws ModelException;
public void setPassword(String password) throws ModelException;
String getPassword() throws ModelException;
public void setContact(String poc) throws ModelException;
String getContact() throws ModelException;
public void setEmail(String email) throws ModelException;
String getEmail() throws ModelException;
public void setPhone(String phone) throws ModelException;
```

```
String getPhone() throws ModelException;
public void setNotes(String notes) throws ModelException;
String getNotes() throws ModelException;
```

The EFSPManager controls access to EFSP objects. This access is typically creation, addition, removal, or retrieval of an EFSP from the system. The manager also provides the ability to validate the username and password pair for a given EFSP. This functionality is defined by the following methods:

```
List getAllEfilingProviders() throws ModelException;
public EFSP createEFSP() throws ModelException;
public void addEFSP(EFSP efsp) throws ModelException;
EFSP getEfilingProvider(Id efspId) throws ModelException;
public void removeEFSP(EFSP efsp) throws ModelException;
public void updateEFSP(EFSP efsp) throws ModelException;
public EFSP findEfilingProviderByUsername(String username) throws ModelEx-
   ception;
public boolean validateEFSP(String username, String password) throws Mod-
elException;
```

3. ID Object Model

Unique IDs are important for any functional computer system. OpenEFM has defined interfaces for an Id and an IdFactory.

An ID object simply requires that any subclasses of the interface extend the Java-comparable class. This ensures that any two implementations of an ID object can be easily compared. No methods are defined in this interface. Requiring a method to access the ID value itself would dictate what form the system ID should be in (i.e., Int, String, etc.); therefore, no assumption is made.

The IdFactory object creates strings that are converted into an ID object. Unique identifiers are also available through the factory. Implementations of the IdFactory should be careful to ensure that proper unique identifiers are correctly being implemented. Only the following two methods are defined by the IdFactory.

```
String getNextId() throws IdDispenserException;
String getUniqueIdentifier();
```

4. Query Object Model

Queries are intended to pass through the EFM. As such, an object has been created to represent them. The query object implements an access interface. This interface defines the methods that can be used to retrieve pertinent information from the query. The methods defined in the interface are listed below:

```
// *** QUERY ***
public  List getCaseListQueryIds();
  public  List getCaseQueryIds();
  public  List getDocumentQueryIds();
  public  String getPolicyQueryId();

  public  String getCourtId(String queryId);
  public  List   getCaselistParticipants(String queryId);
  public  String getCaseTrackingId(String queryId);
  public  String getCaseDocketId(String queryId);
  public  String getCaseSubsetText(String queryId);


 // *** RESPONSE ***
public  List   getCaseListResponseIds();
  public  List   getCaseResponseIds();
  public  String getPolicyResponseId();
  public  List   getDocumentResponseIds();
  public  List   getCaseTrackingIds(String caseListResponseId);

  public  String getCaseStatus(String responseId, String caseTrackingId);
  public  List   getDefendantPartyPerson(String responseId, String caseTrack-
ingId);
  public  List   getProsecutionAttorney_PersonName(String responseId, String
caseTrackingId);
  public  String getProsecutionAttorney_JudicialOfficialBarID(String responseId,
String caseTrackingId);
  public  String getDocumentSubmission_DocumentTitleText(String responseId);
  public  String getPolicyReferenceURI();
```

5. Legal XML Object Model

The LegalXML object model defines the data elements needed for the system to handle the creation, handling, acceptance, and validation of LegalXML filings. This includes LxmlFiling, InternalLxmlFiling, LxmlResponse, and LxmlFilingAttachment.

LxmlFiling is a small object which holds the actual XML data that represents the electronic legal filing. This interface defines the following methods, allowing for setting the LegalXML content of the object and retrieving the LegalXML content:

```
public void setContent(String content);
public String getContent();
```

InternalLxmlFiling objects extend the LxmlFiling object. They add information retrieval capability to the filing. Methods are also defined that maintain the state of the filing in OpenEFM. The possible states of the InternalLxmlFiling are pending, accepted, and rejected. The InternalLxmlFiling interface defines a large number of data accessing methods. These methods retrieve information from the actual XML filing. This is implemented through standard XPath queries. This allows for the XPath queries to change as the underlying XML standards are modified, without needing to reimplement the actual classes.

For the sake of brevity, the methods defined in the InternalLxmlFiling interface are listed in APPENDIX B.

The LxmlFilingManager controls access to LxmlFiling objects stored in the system. This includes creation, addition, removal, and retrieval functionality. Various methods are defined to allow for different search parameters. The defined methods are as follows:

```
List getAllFilings() throws ModelException;
List getPendingFilings() throws ModelException;
List getRejectedFilings() throws ModelException;
List getAcceptedFilings() throws ModelException;
List getAllFilingsByCourt(Id courtId) throws ModelException;
List getAllFilingsByEfilingProvider(Id efspId) throws ModelException;
List getAllFilingsByCourtAndEfilingProvider() throws ModelException;
List getFilingsOlderThan(Date date) throws ModelException;
public InternalLxmlFiling createLxmlFiling() throws ModelException;
public void addFiling(InternalLxmlFiling filing) throws ModelException;
InternalLxmlFiling getFiling(Id filingId) throws ModelException;
public void removeFiling(Id filingId) throws ModelException;
public void acceptFiling(Id filingId) throws ModelException;
public void rejectFiling(Id filingId) throws ModelException;
```

C. TRANSPORT MODULE

The transport module objects handle all tasks related to communications. This includes transceivers that control incoming and outgoing communication, message wrappers that model the protocol specific formats and CMS or DMS adapters. The transceivers and message wrappers are transport layer protocol specific. Implementations exist for HTTP, SOAP and ebXML.

1. Transceiver Objects

Transceivers represent communications with external systems occurring. These occurrences may be incoming or outgoing. Examples of these communications are filing submissions, queries passing through the system, requests for court policy and connecting to an external CMS.

A basic transceiver interface defines methods for setting the destination and callback URIs, and also for transmitting a message object. This file is located in the source tree at:

```
src/com/counterclaim/openefm/transport/transceiver/Transceiver.java
```

Transceiver implementations exist for ebXML, SOAP and HTTP.

2. Message Wrapper Objects

Message wrapper objects represent the format of the actual messages that will be transmitted via different protocols. Currently there are implementations for ebXML and SOAP. The message wrapper objects typically hold the actual data that is going to be transmitted. This data includes the XML instance documents containing the court filing information. Data such as document attachments and conversation IDs are also contained in these objects.

3. Adapter Objects

Adapter objects contain the actual submitFiling() and submitQuery() metbod calls.

```
public LxmlResponse submitFiling(InternalLxmlFiling filing);
public QueryResponse submitQuery(Query query);
```

These methods take the model object representing filings and queries as input. They use this data by encapsulating it in a message wrapper object and sending it via a transceiver.

D. SECURITY MANAGER MODULE

The security manager module is in charge of verifying the authenticity of an EFSP submitting a filing. It is invoked when a new filing arrives at the EFM through the FilingTransceiver. The FilingTransceiver object passes the filing off to the OpenEFMController to be handled appropriately. The OpenEFMController passes the filing off to the security manager to ensure that it has been properly established and has filing permission with the EFM.

Currently, the security manager interface defines a single method:

```
SecurityViolation acceptFiling(InternalLxmlFiling filing);
```

The implementation of the security manager module that is included with OpenEFM is a simple username- and password-based authentication layer. The SimpleSecurityManager object implements this interface. The SimpleSecurityManager validates the EFSP username and password, contained within the InternalLxmlFIling object, with the EFSPManager object.

E. LXML VALIDATOR MODULE

The LegalXML validator module is used to ensure that LegalXML filings entering the system are well formed and conform to the correct DTD or schema. One method is defined in this module interface:

```
ValidationFault[] validateFiling(LxmlFiling filing);
```

ValidationFault is a class defined in the LxmlValidator module that records data regarding a problem with parsing or validating an XML filing. Currently, there is a SimpleLxmlValidator implementation and a SchemaLxmlValidator that implements the LxmlValidator interface. SchemaLxmlValidator uses SAX to validate a filing.

F. AUDIT LOGGER MODULE

The audit logger module is designed to provide simple method definitions representing system actions that require log entries. Currently, two actions have been identified as requiring log entries. These actions are a filing being received and a filing failing a security check. The method definitions are as follows:

```
void filingReceived(InternalLxmlFiling filing);
void filingFailedSecurityCheck(InternalLxmlFiling filing, SecurityViola-
   tion violation);
```

The list of actions requiring log entries will likely expand to include actions such as the acceptance or rejection of a filing. Also, the sending and receiving of queries and responses will probably require log entries. The method definitions will likely expand as the project continues.

Currently, OpenEFM provides a file system logger. This logger is based on Log4j, an open source logging library provided and maintained by Apache Software Foundation. The file system logger writes out log files to a local disk.

A new AuditLogger implementation could be easily created by a court that would like to log actions in a different manner, such as to a database or sent to another system over a network layer. The implementation to be loaded into OpenEFM at runtime is determined by the entry in the XML-based configuration file.

G. ID FACTORY MODULE

The ID Factory module is part of the general object model hierarchy of OpenEFM. The factory is required to provide the next usable ID and unique identifier in OpenEFM. It also supplies SOAP conversation IDs. Currently, OpenEFM provides an implementation of this module based on a millisecond timestamp. If two IDs are requested in the same millisecond then the millisecond count is incremented to provide a unique number. This implementation fulfills all requirements listed in the OXCI EFM Requirements Document. However, if the millisecond timestamp method should prove unacceptable at some future date, it can be easily replaced in the system.

H. BILLING MODULE

The billing module is used to process electronic fund transfer (EFT) payments. Currently, this module defines only one method:

```
BillingResponse proccessFees(InternalLxmlFiling filing);
```

It is assumed that the required billing information, fees, credit card information, etc., are stored in the InternalLxmlFiling object. The BillingResponse object represents various possible responses an EFT transaction might produce. This includes authorization and reference numbers, as well as possible error codes used by the EFT merchant.

OpenEFM have been configured to work with two separate EFT merchants, the most popular of which being VeriSign, Inc. A billing module implementation that interfaces with the VeriSign EFT system is provided with OpenEFM.

This meets the requirements regarding EFT as defined by OXCI.

IV. DEVELOPMENT REQUIREMENTS

IV. DEVELOPMENT REQUIREMENTS

The vast majority of the unmet requirements outlined in the OXCI EFM Software Requirements document revolved around the schema implementations of the Query and Response API, as well as the court filing and court policy definitions. Due to these relationships, unmet requirements were grouped according to the individual underlying requirement dependencies. One underlying dependency shared by all unmet requirements was the ability to utilize the XML Schema by OpenEFM. The other defined dependencies are court filing, query/response, and court policy issues. Billing issues will also be discussed.

A. W3C XML SCHEMAS AND COURT FILING ISSUES

Previously, OpenEFM had been configured to accept LegalXML filings based on a DTD. To comply with requirements 1 and 2 of subsection III.A. in the OXCI EFM Software Requirements document, filings based on the W3C Schema-based Court Filing specification were needed to be accepted.

In order to handle filings as schemas, two functional requirements were met. First, the implementation of the InternalLxmlFiling object needed to be able to access information in an XML filing based on a schema. Second, the LegalXML validation module will need to be able to validate against a schema. Both of these functional requirements were met prior to completion of the project.

1. InternalLxmlFiling

The InternalLxmlFiling object, as it was implemented, was not expected to need to be modified in any way to allow access to information in any XML filing.Access to data stored in the XML filing is granted by running XPath queries against a Document Object Model (DOM) object hierarchy. The DOM object hierarchy should have no problem parsing a well-formed XML document in any format. As the XML filing formats change, the XPath queries were believed to be the only information in need of updating in order to allow access to data in a new XML format. This proved not to be entirely correct. The location of much data was shifted from the XML specifications into the SOAP and ebXML transport layer. This required that the fundamental structure of the InternalLxmlFiling change on some level. Once these changes were made all filing data was accessible. The methods relying mainly on XPath queries are listed in APPENDIX B.

2. Validation Module

The SimpleLxmlValidation implementation provided with OpenEFM relies on an underlying DTD. The validation is accomplished by using a Simple API for XML (SAX) to parse the filing. Since SAX is capable of validating W3C Schema-defined files, the changes required were not dramatic. The changes needed to allow the validation were to add a schema location attribute to the validation module, much like the current DTD location attribute. Also, a few schema-specific SAX properties were needed to be set by the parser. The end result was a new SchemaLxmlValidator object.

B. QUERY AND RESPONSE ISSUES

A vast majority of the original unmet requirements outlined in the OXCI EFM Software Requirements document specifically involved the bidirectional Query and Response capability. The common requirements that all Query and Response issues are dependent upon are requirement number 6 in subsection IV.A. and requirement number 3 in subsection III.A.

Requirement 3 in subsection III.A. of the requirements document specifies that all Query and Response messages will validate against the W3C Schema definition of the Query and Response API.

Requirement 6 in part subsection IV.A. of the requirements document specifies that Query and Response messages in the form of the Query and Response specifications will be required to arrive from the CMS and be sent to the EFSP, as well as arrive from the EFSP and be sent to the CMS. The Transceiver module of OpenEFM will define methods for handling these actions. The transport module was extended to meet these requirements. It contains objects capable of representing a transport protocol message, a transceiver to transmit the message, and a CMS or DMS adapter to determine when queries should be passed along.

Now that the Query and Response functionality is implemented in OXCI EFM, the following list of requirements have been met:

- Requirements 2 and 3 in subsection I.A.
- Requirements 1 and 2 in subsection I.B.
- Requirements 1 and 2 in subsection I.C.

Requirement number 2 of subsection III.B. states that the application's SOAP transmission will need to be extended with ebXML. The SOAP interfaces will be made available through the implementation of the Query and Response API, specifically in the Transport module. These SOAP implementations utilize ebXML in order to meet this requirement.

C. COURT POLICY ISSUES

Previously, OpenEFM did not integrate with any form of court policy. Requirement number 3 in subsection III.A. in the requirements document requires that the Court Policy schema be used to provide information to the EFM.

To implement the usage of a court policy XML file, a new module was added to OpenEFM. Specifically, a new object of type CourtPolicy was defined.

Requirement number 2 in subsection II.E., states that the administrator of OXCIEFM will need the ability to access the Court Policy XML file via a provided URL location of the file. Also, OXCI EFM will need to be able to provide the ability to host the Court Policy XML file locally, at a Web-accessible location. This was accomplished by the Court Policy module configuration attributes.

Court Policy Module Attributes

1. Location

The first data element, the location of the hosted Court Policy XML file, is required. This will be a standard HTTP address to the direct location of the XML file.

2. FileLocation

This attribute is optional. If the Court Policy XML file resides on the local file system, it can be addressed here.

3. HostLocation

This attribute is optional. This attribute would be used when the Court Policy XML file resides on the local file system, and the administrator would like OpenEFM to host the file. The HostLocation will be the Web address the administrator would like OpenEFM to host the file at.

4. HostFileName

This attribute is optional. It can be used to specify the name of the Court Policy XML file when the file is hosted by OpenEFM.

5. Schema

This attribute is required. It is the location of the Court Policy schema on the local file system. The schema file will be used to validate Court Policy XML files when they are processed by the system.

The CourtPolicy object will need to define methods used in accessing the data containing a CourtPolicy XML file.

D. BILLING INFORMATION ISSUES

The billing information associated with a filing includes data such as credit card numbers and expiration dates. This is sensitive information. Because of how sensitive this data can be, OXCI would like to keep this data separate from the Court Filing specification. Instead, the specification will reference the data being stored elsewhere.

In order to meet requirement 5 in subsection IV.A. of the requirements document, the billing information is included in the ebXML extensions of SOAP. This utilizes the preexisting functionality of ebXML as it was intended.

V. GRAPHICAL USER INTERFACE

V. GRAPHICAL USER INTERFACE

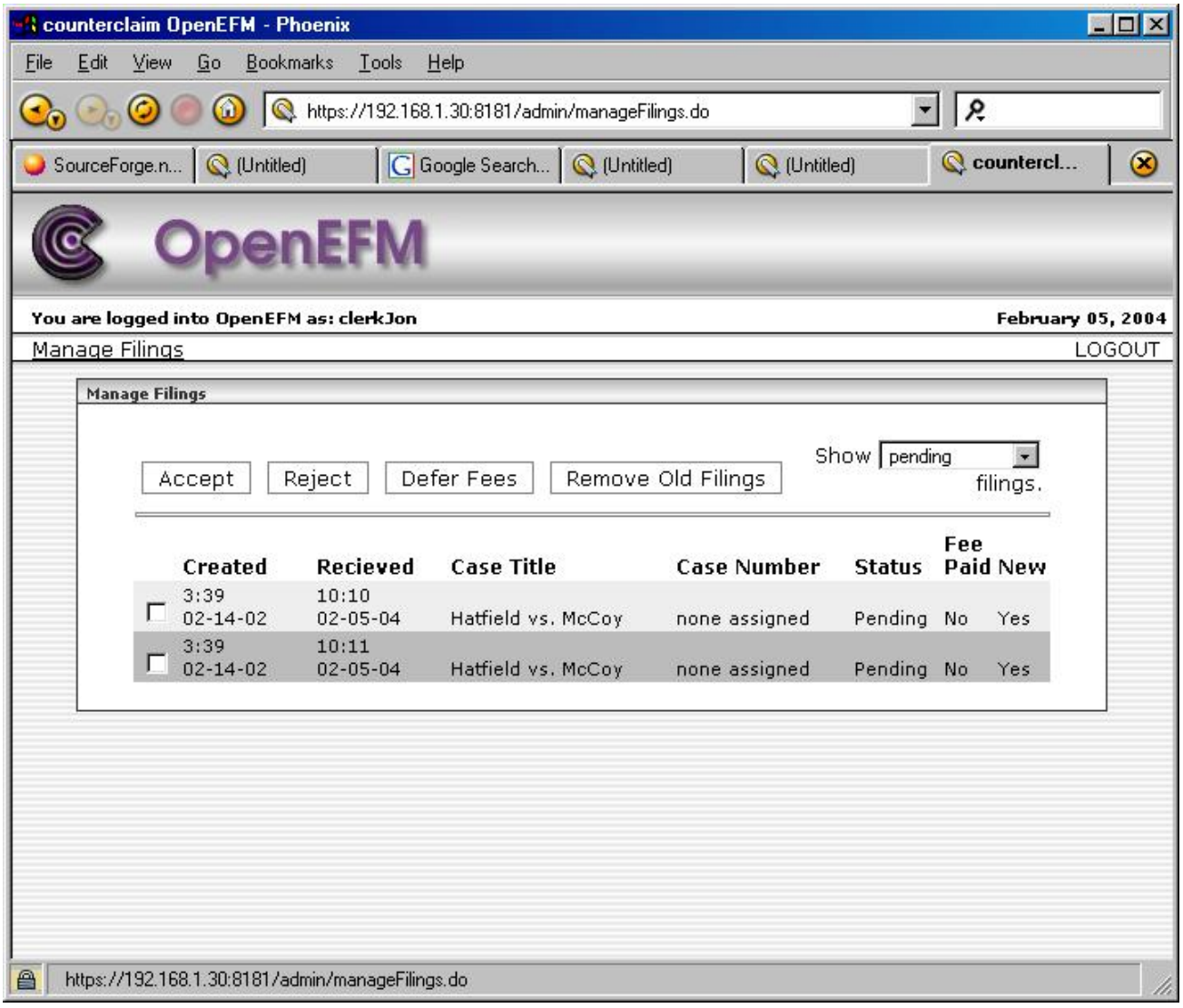
Currently, several user interface elements are implemented in OpenEFM. The elements enable the required actions by users. These actions include logging in and out of the system, reviewing and processing filings, and managing users and EFSPs.

A. LOGIN

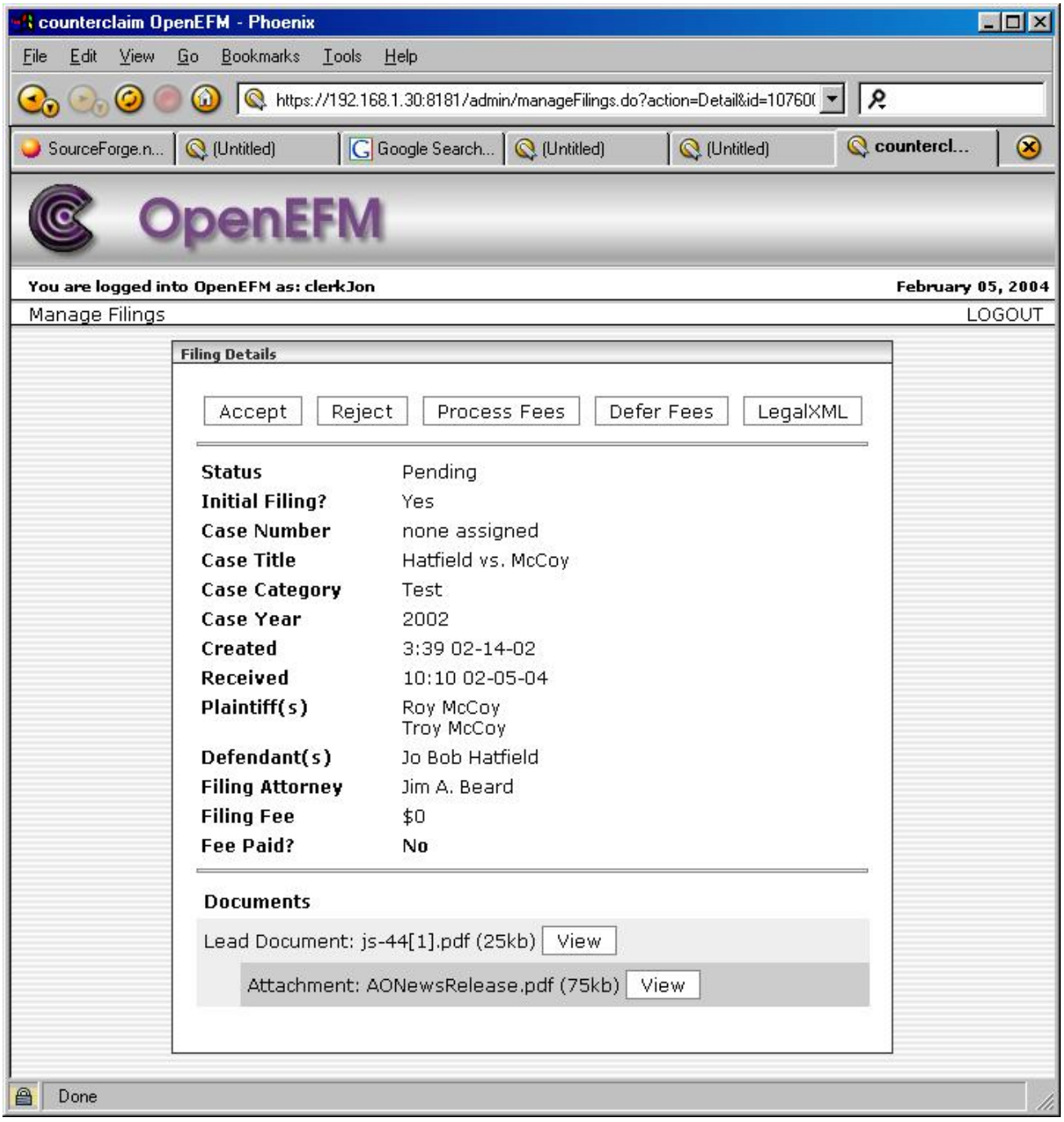
All users of the system, administrators and court clerks alike, will be required to log into a central Web site.

B. MANAGE FILINGS

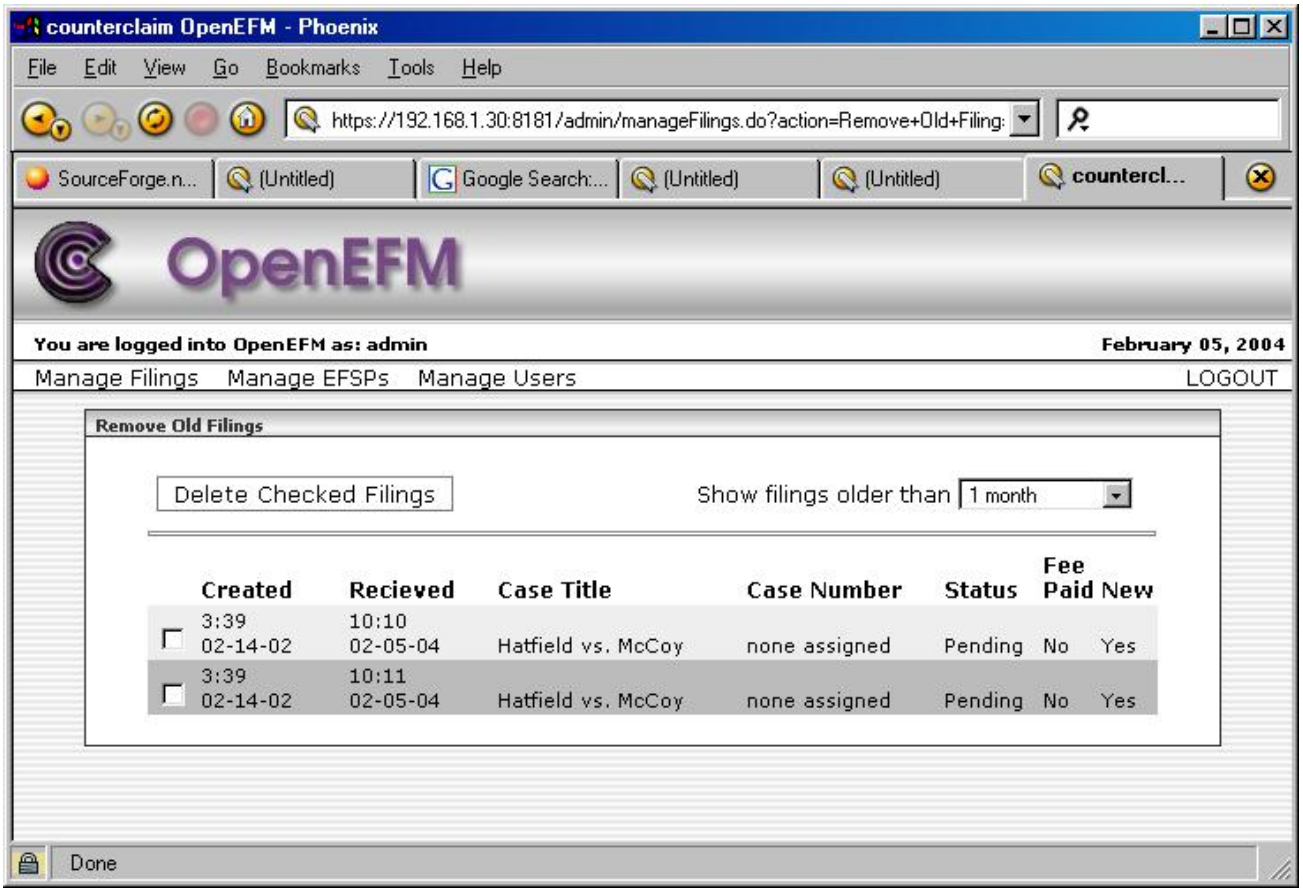
Both administrative accounts and clerk accounts have the ability to manage filings in the system. This includes listing the current filings in the system, viewing the details of an individual filing, and removing old filings from the system.



C. FILING DETAIL

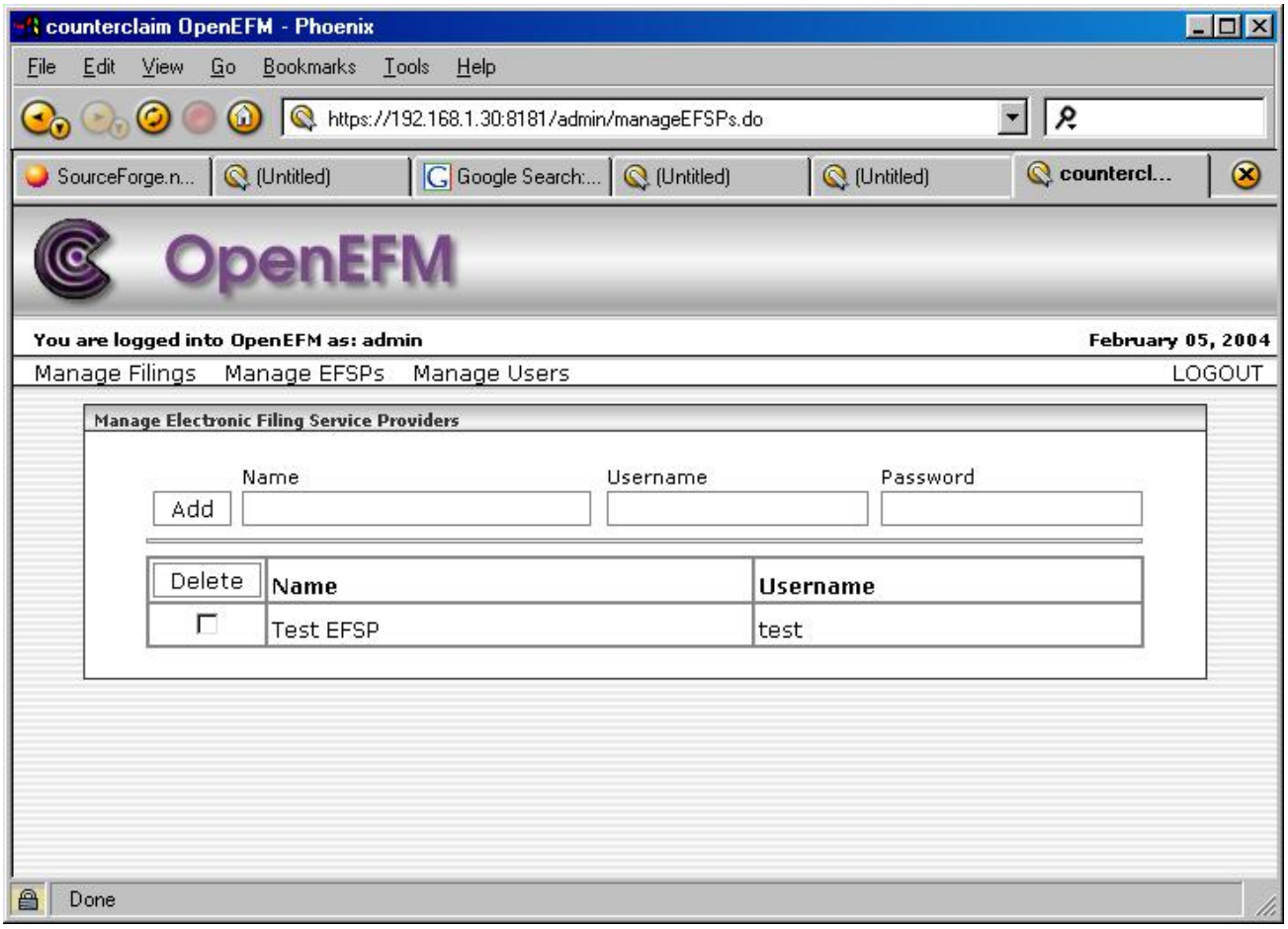


D. DELETION OF AGED FILINGS

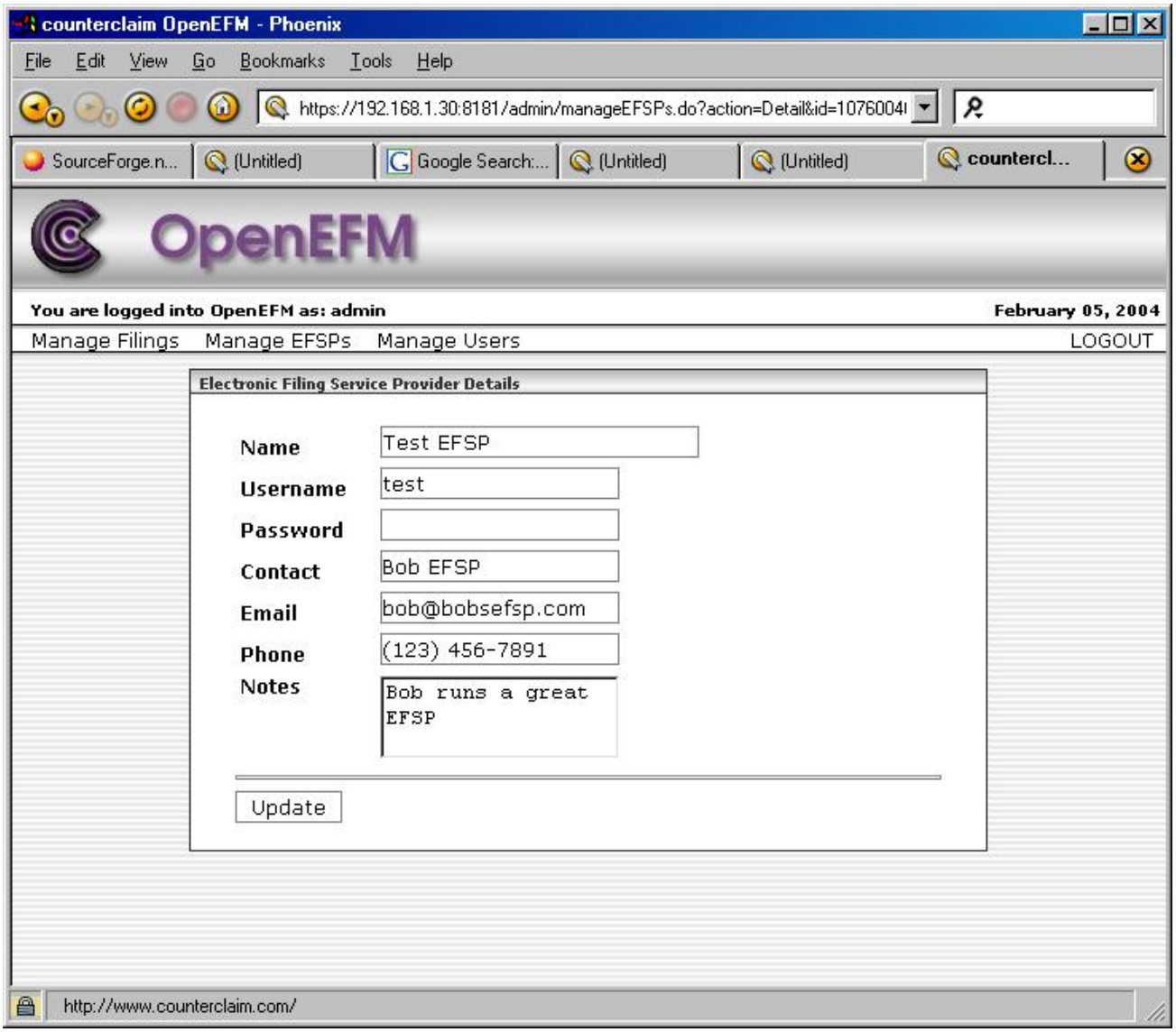


E. MANAGE EFSPS

System administrators are capable of managing EFSPs that are able to communicate with OpenEFM. The user interface elements that enable the EFSP management allow for addition of new EFSPs and the ability to modify information about existing EFSPs.

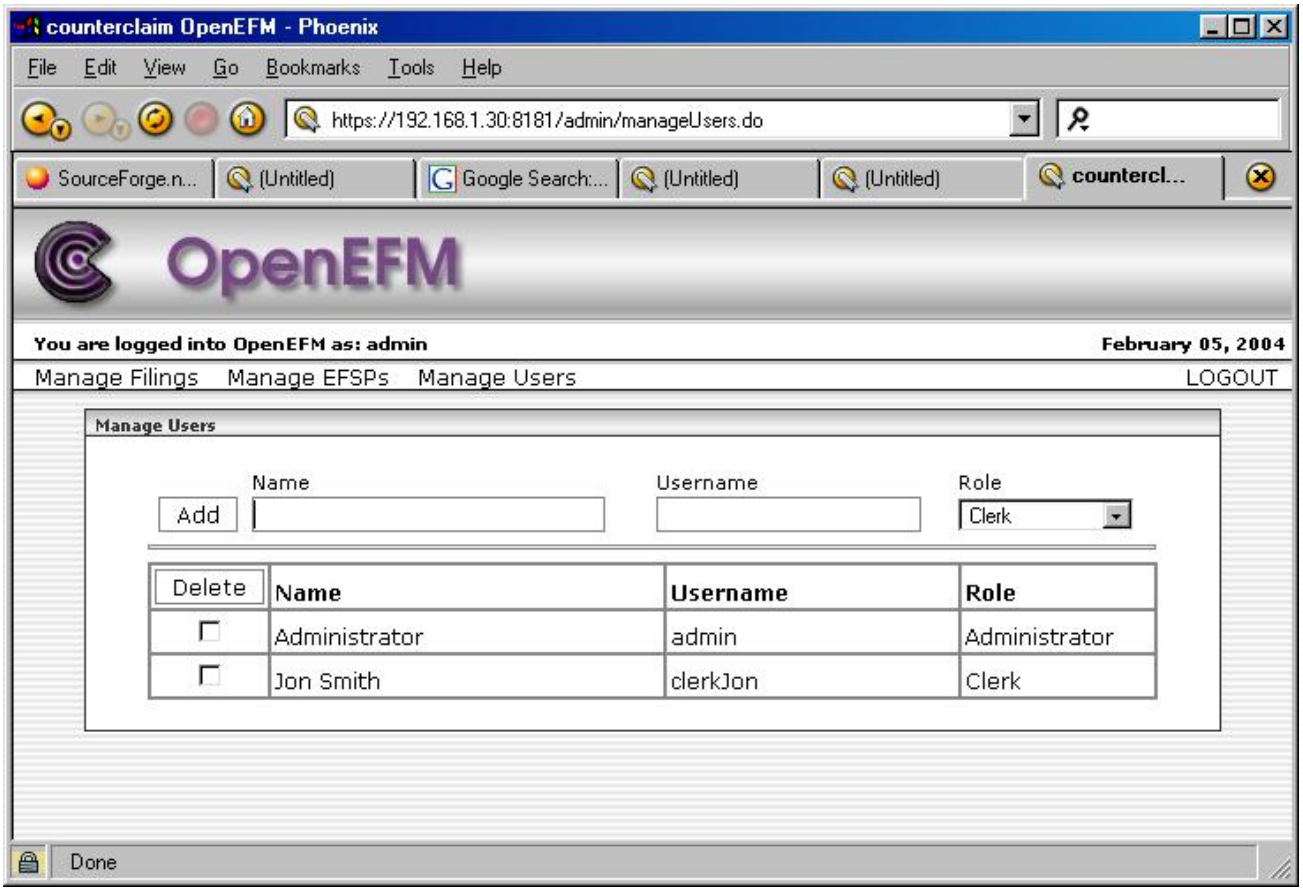

counterclaim OpenEFM - Phoenix
File Edit View Go Bookmarks Tools Help
https://192.168.1.30:8181/admin/manageEFSPs.do
SourceForge.n...
(Untitled)
Google Search:...
(Untitled)
(Untitled)
countercl...
OpenEFM
You are logged into OpenEFM as: admin
February 05, 2004
Manage Filings Manage EFSPs Manage Users
LOGOUT
Manage Electronic Filing Service Providers
Name
Username
Password
Add
Delete
Name
Username
Test EFSP
test
Done

F. EFSP DETAIL

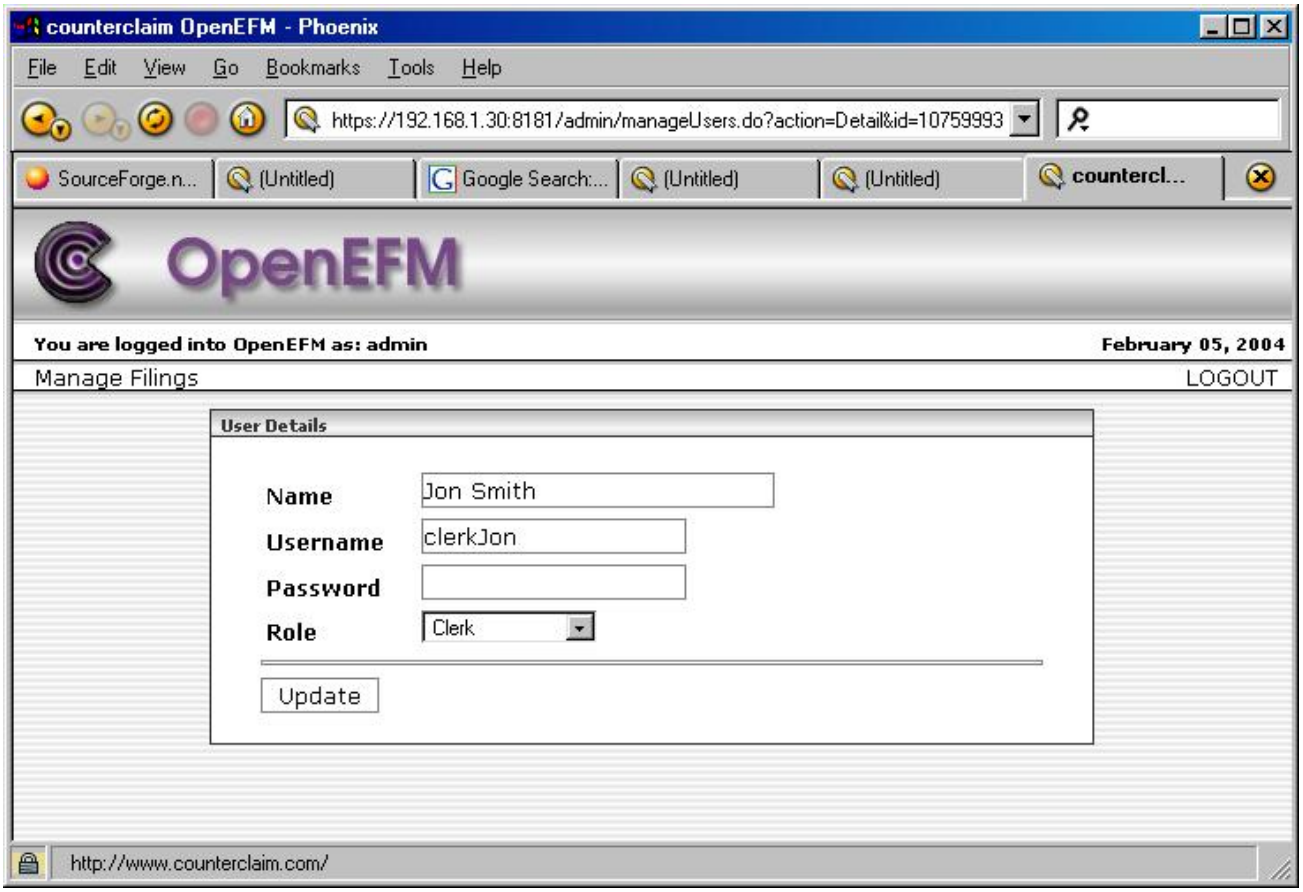


G. MANAGE USERS

System administrators are also capable of controlling user account access to OpenEFM. This includes adding and removing users from the system. Information stored by the system, regarding the user, is also editable.


counterclaim OpenEFM - Phoenix
File Edit View Go Bookmarks Tools Help
https://192.168.1.30:8181/admin/manageUsers.do
SourceForge.n... (Untitled) Google Search:... (Untitled) (Untitled) countercl...
OpenEFM
You are logged into OpenEFM as: admin
February 05, 2004
Manage Filings Manage EFSPs Manage Users LOGOUT
Manage Users
Name
Username
Role
Add
Clerk
Delete
Name
Username
Role
Administrator
admin
Administrator
Jon Smith
clerkJon
Clerk
Done

H. USER DETAIL



Currently, the user interface elements in OpenEFM allow for all tasks defined by the OXCI EFM Requirements Software document that require a user interface to be completed. There is no plan to modify the existing user interface in order to comply with any aspects of the OXCI EFM Software Requirements document.

APPENDIX A
GLOSSARY

GLOSSARY

API	Application Program Interface
CMS	Case Management System
DMS	Document Management System
DOM	Document Object Model
DTD	Document Type Definition
ebXML	Electronic Business xtensible Markup Language
EFM	Electronic Filing Manager
EFSP	Electronic Filing Service Provider
EFT	electronic fund transfer
HTTP	HyperText Transfer Protocol
JAXB	Java Architecture for XML Binding
OXCI	Open XML Court Interface
SAX	Simple API for XML
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
XML	xtensible Markup Language

APPENDIX B
DEFINED METHODS OF THE INTERNAL LXML FILING INTERFACE

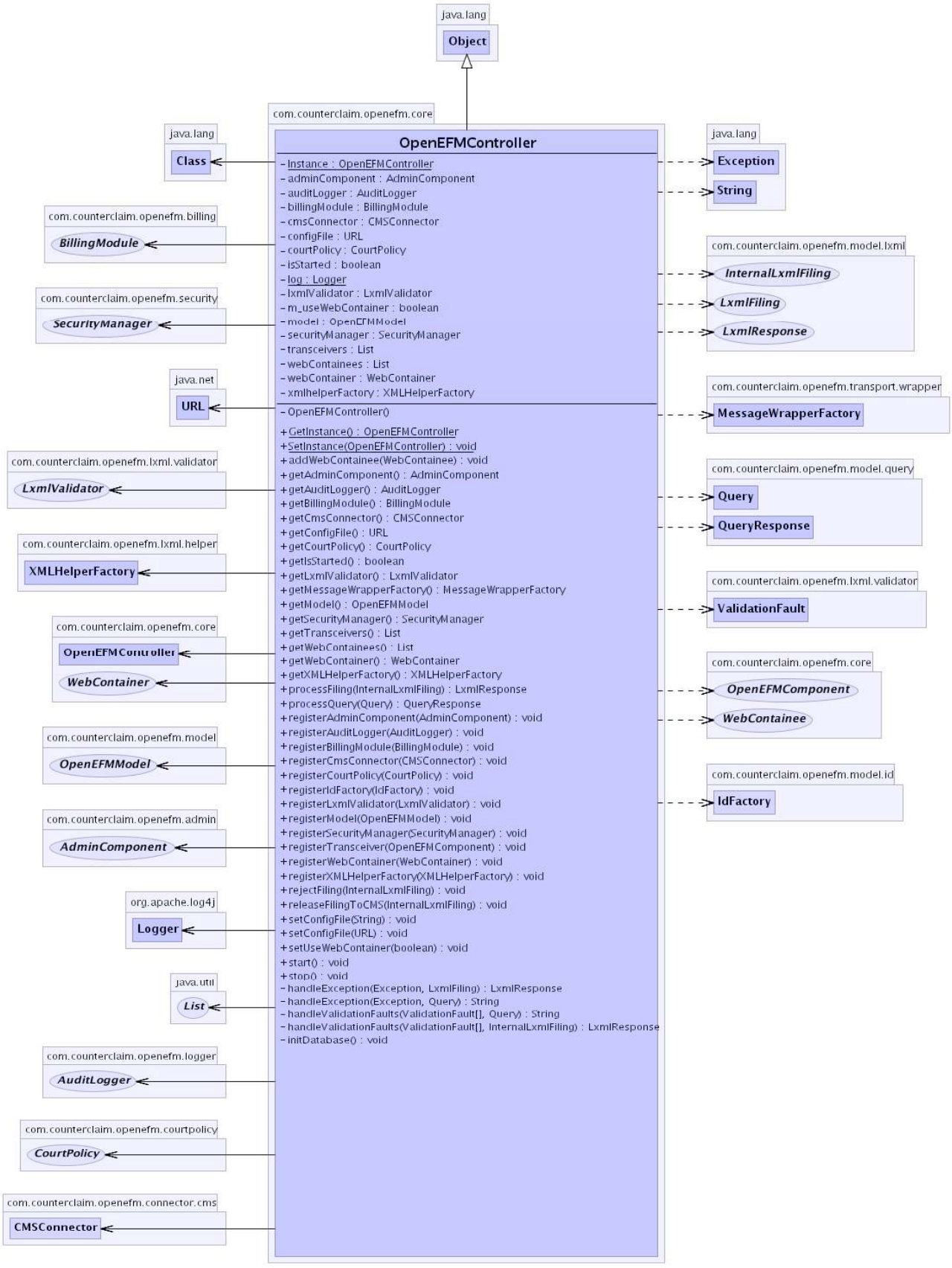
DEFINED METHODS OF THE INTERNAL LXML FILING INTERFACE

```
Id getId() throws ModelException;
void setId(Id id) throws ModelException;
 eceive initXPath();
 eceive isQueriable();
List queryXPath(String query);
public String getMessageIdentification();
public Date getCreation();
public String getLeadDocumentId();
public List getDocumentActorReferences(String docId);
public Date getDocumentSubmitted(String docId);
public String getDocumentTitle(String docId);
public String getDocumentType(String docId);
public String getDocumentContent(String docId);
public String getDocumentContentId(String docId);
public Integer getDocumentContentSize(String docId);
public String getDocumentContentMimeType(String docId);
public String getDocumentContentEncoding(String docId);
public String getDocumentContentHref(String docId);
public List getDocumentAttachementIds(String docId);
public List getToFullNames();
public List getToURIs();
public List getFromFullNames();
public List getFromURIs();
public List getMemos();
public List getActorIds();
public List getActorIdsWithRole(String role);
public String getActorFullName(String actorId);
public String getActorFirstName(String actorId);
public String getActorMiddleName(String actorId);
public String getActorLastName(String actorId);
public String getActorRoleName(String actorId);
public List getActorRoleWithIds(String actorId);
public String getActorTitle(String actorId);
public String getActorEmail(String actorId);
public String getActorPhone(String actorId);
public List getActorAddressLines(String actorId);
public String getActorAddressCity(String actorId);
public String getActorAddressCounty(String actorId);
public String getActorAddressState(String actorId);
public String getActorAddressZip(String actorId);
public String getActorAddressCountry(String actorId);
public List getFilingInformationIds();
public String getFilingFee(String filingInformationId);
public String getCreditNumber(String filingInformationId);
public String getCreditExpDate(String filingInformationId);
public String getCreditOwner(String filingInformationId);
public List getCreditAddressLines(String filingInformationId);
public String getCreditAddressCity(String filingInformationId);
public String getCreditAddressCounty(String filingInformationId);
public String getCreditAddressState(String filingInformationId);
public String getCreditAddressZip(String filingInformationId);
public String getCreditAddressCountry(String filingInformationId);
public String getFilingFeeType(String filingInformationId);
public String getCourtLocationQualifier(String filingInformationId);
```

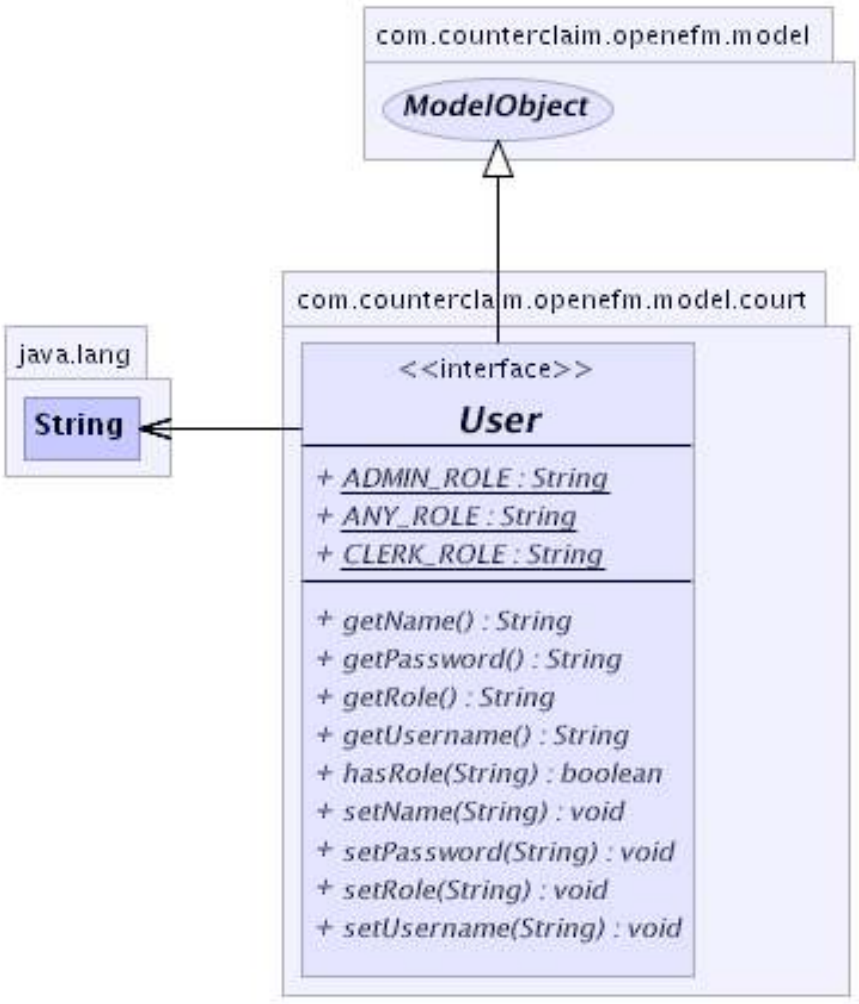
```
public String getCourtType(String filingInformationId);
public String getCourtName(String filingInformationId);
public Boolean isNewCase(String filingInformationId);
public String getFullCaseNumber(String filingInformationId);
public String getCaseTitle(String filingInformationId);
public String getCaseCategory(String filingInfoId);
public String getCaseYear(String filingInformationId);
public String getFiledByActorId(String filingInformationId);
public String getFilingPartyEmail(String filingActorRef);
public void setEFSP(EFSP efsp) throws ModelException;
EFSP getEFSP();
public void setCourt(Court court) throws ModelException;
Court getCourt();
public void setStatus(String status) throws ModelException;
String getStatus();
 eceive isPending();
 eceive isAccepted();
 eceive isRejected();
public void setFeePaid( eceive feePaid);
 eceive isFeePaid();
public void setDiffered( eceive dif);
 eceive isDiffered();
public void setReceived(Date  eceived) throws ModelException;
Date getReceived();
void setContext(FilingContext context);
FilingContext getContext();
```

APPENDIX C
CLASS DIAGRAMS

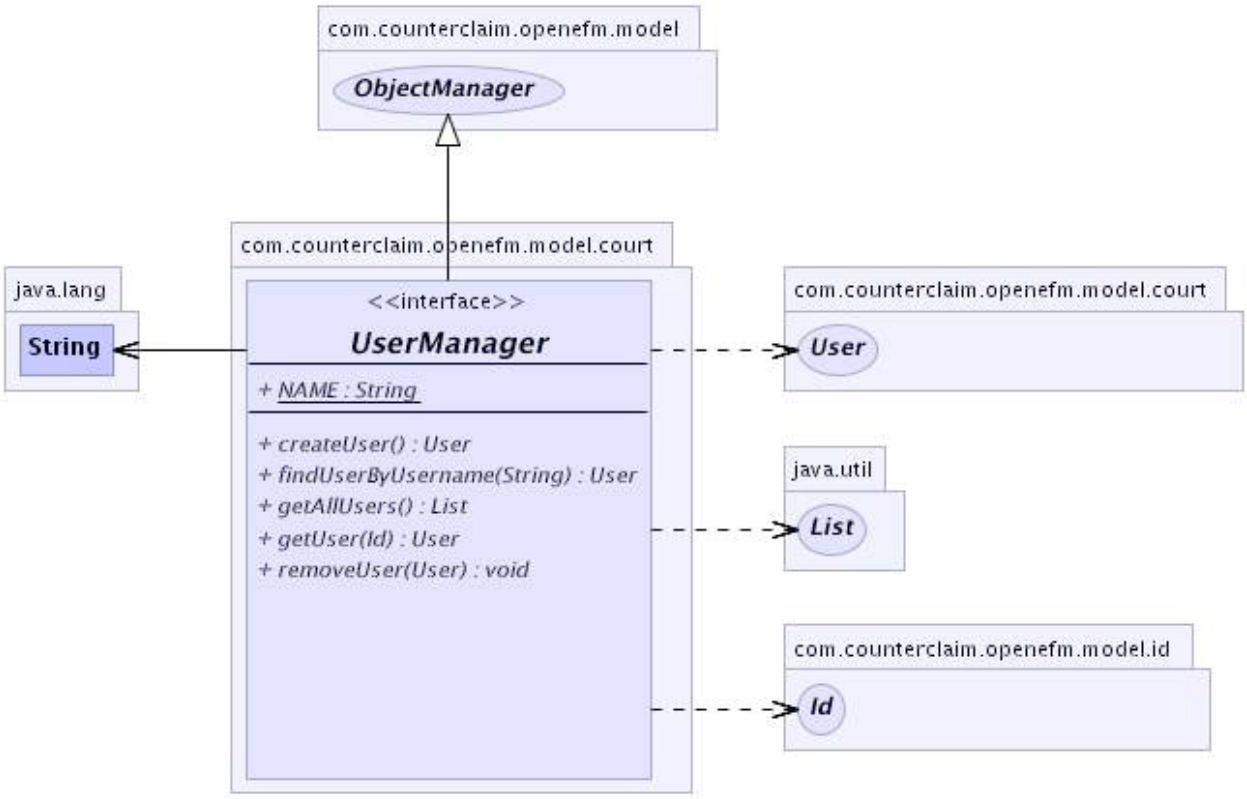
A. OpenEFMController Object



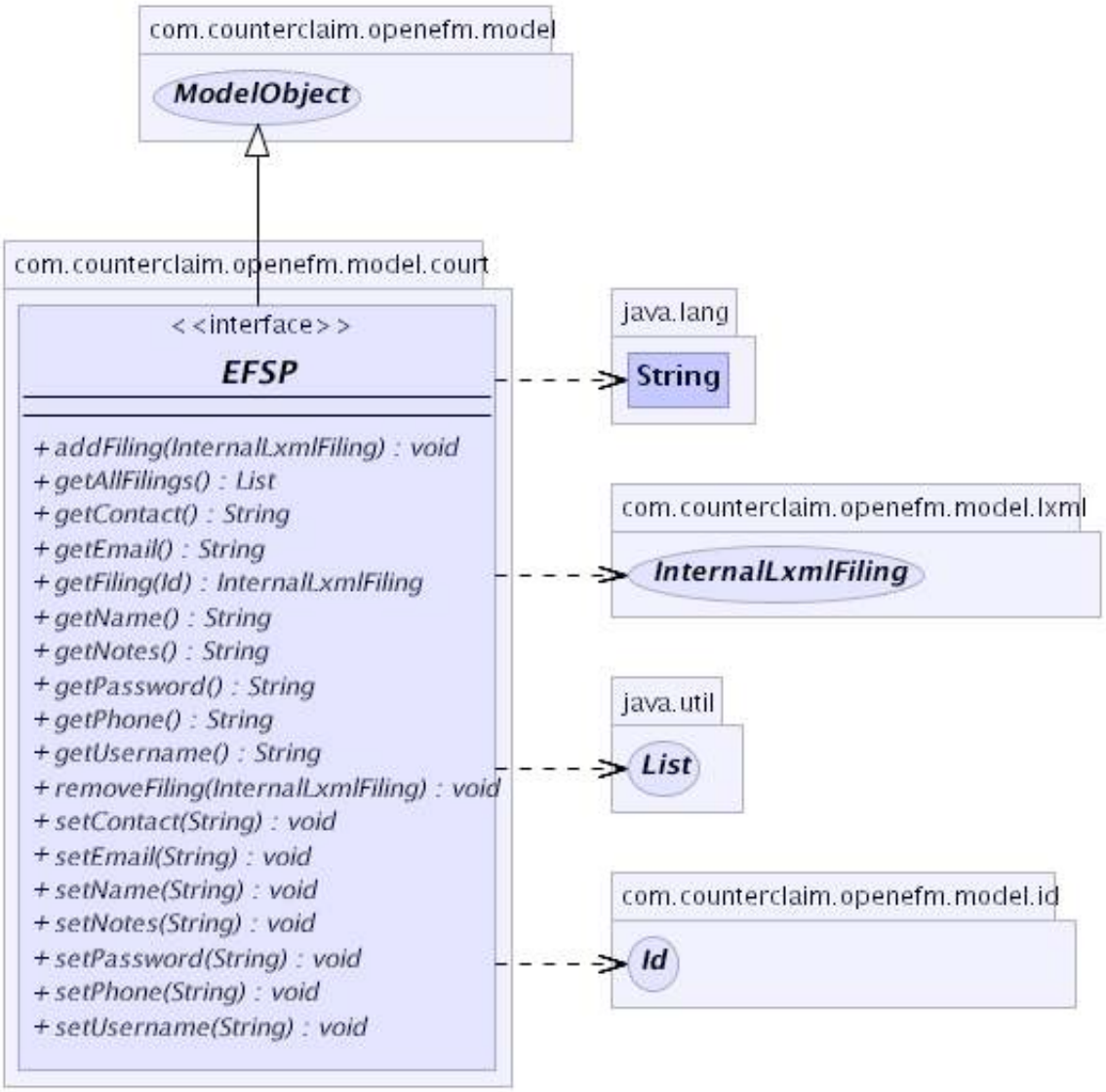
B. User Object



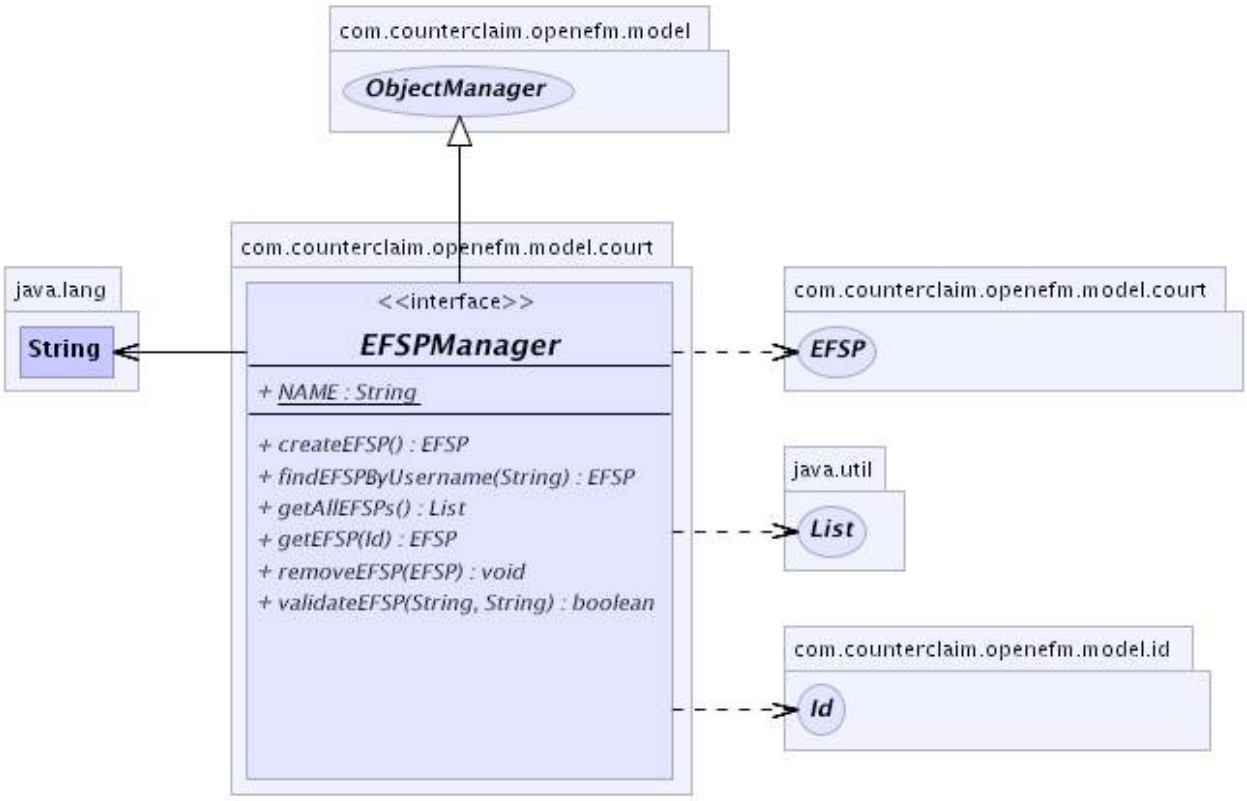
C. UserManager Object



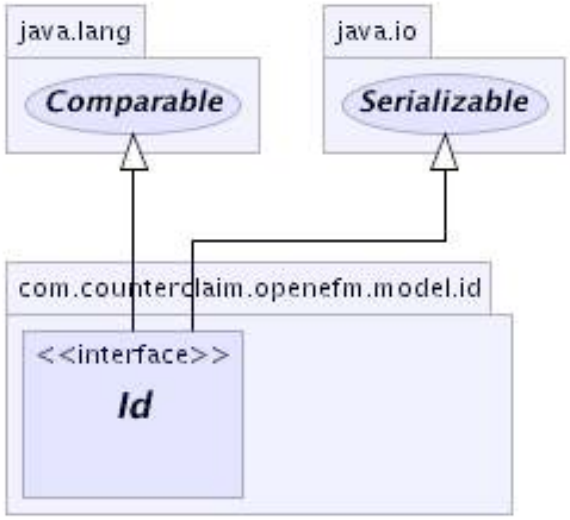
D. EFSP Object



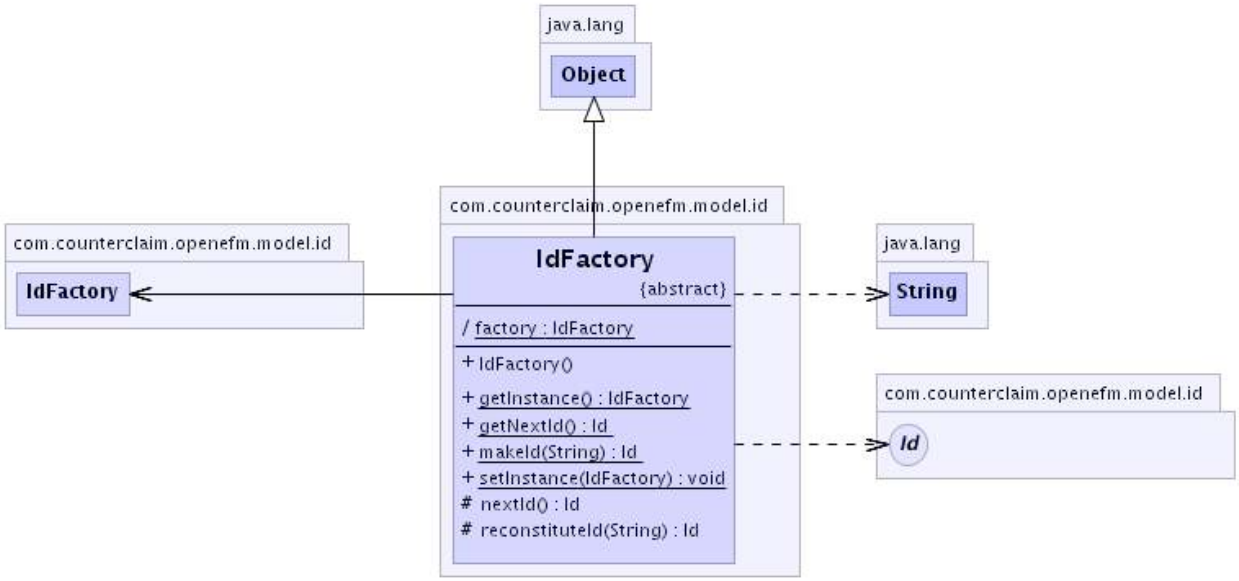
E. EFSPManager Object



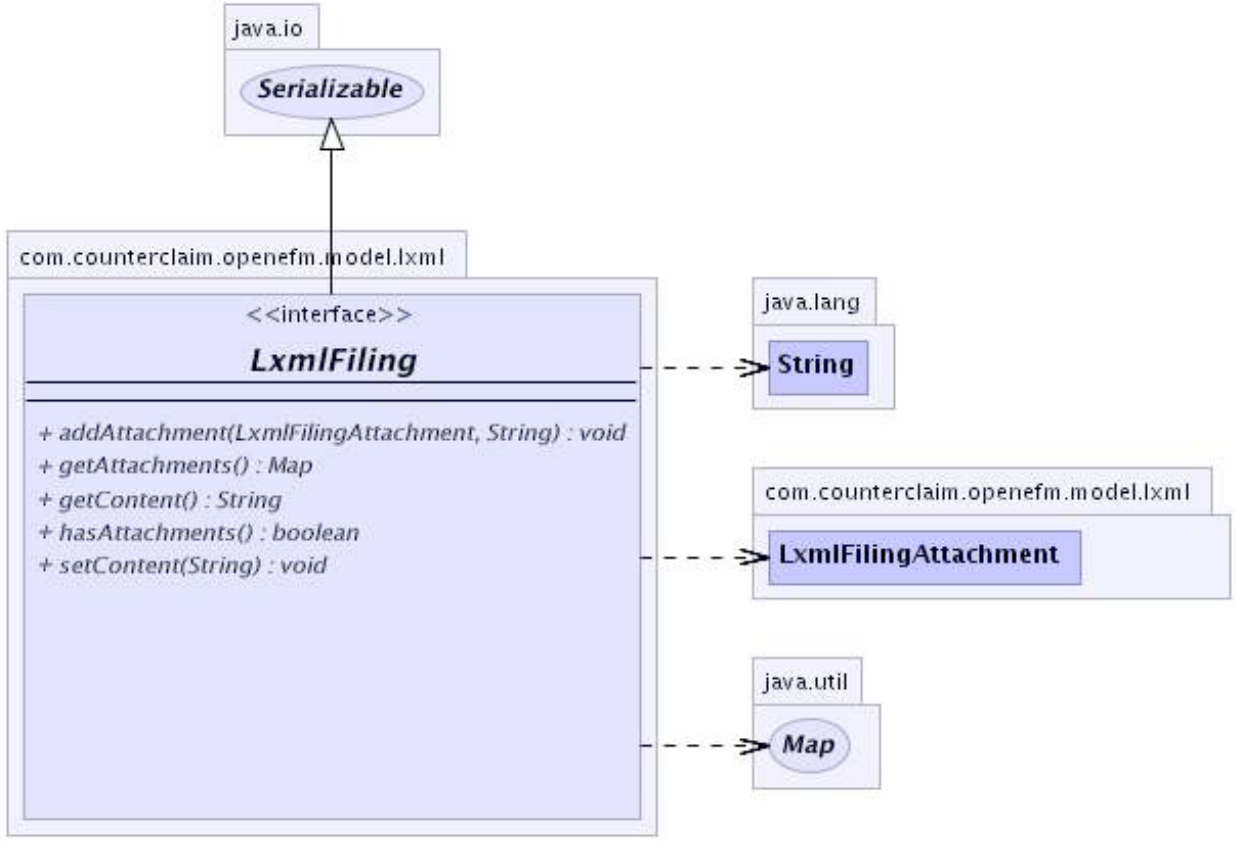
F. ID Object



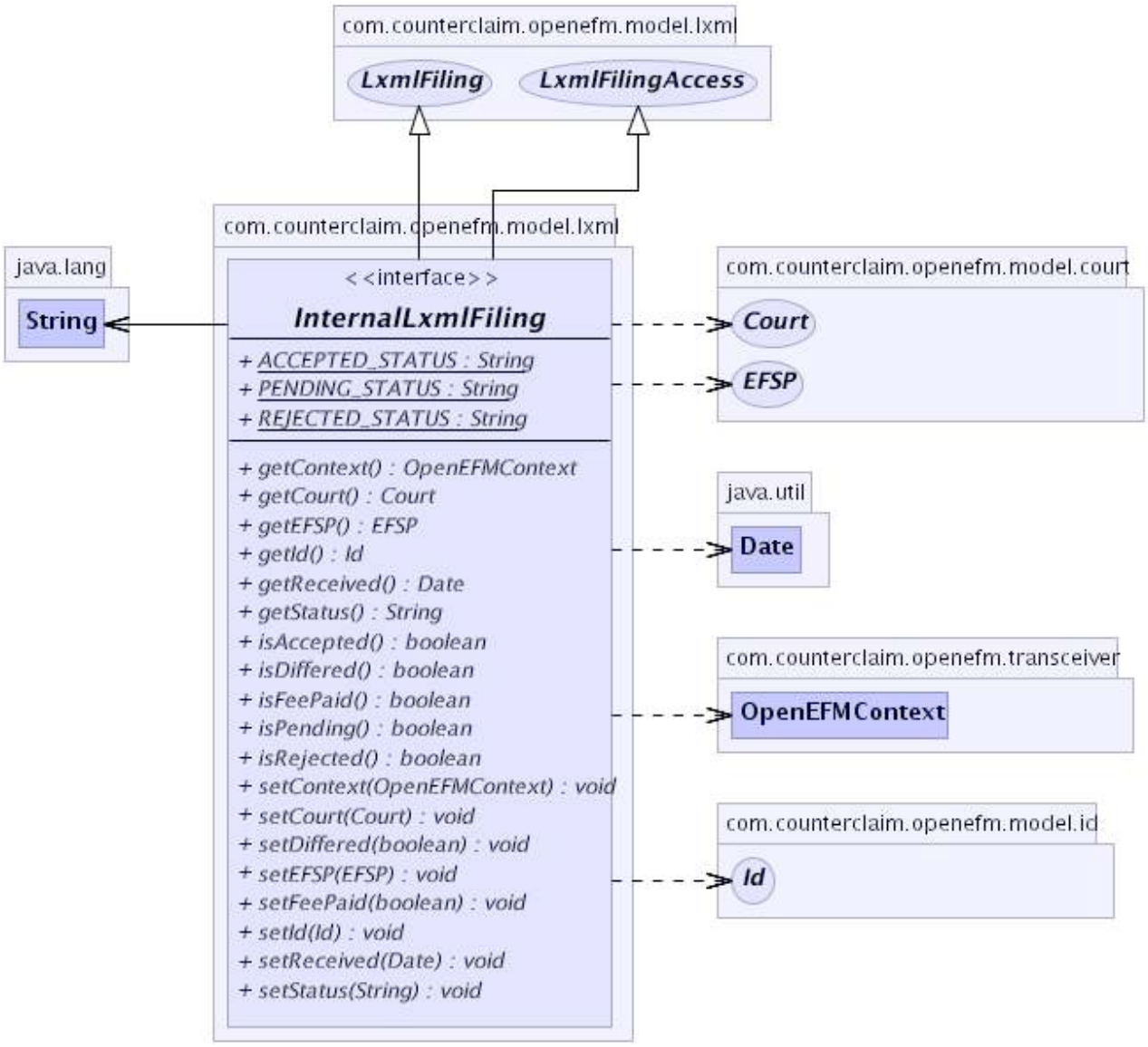
G. IDFactory Object



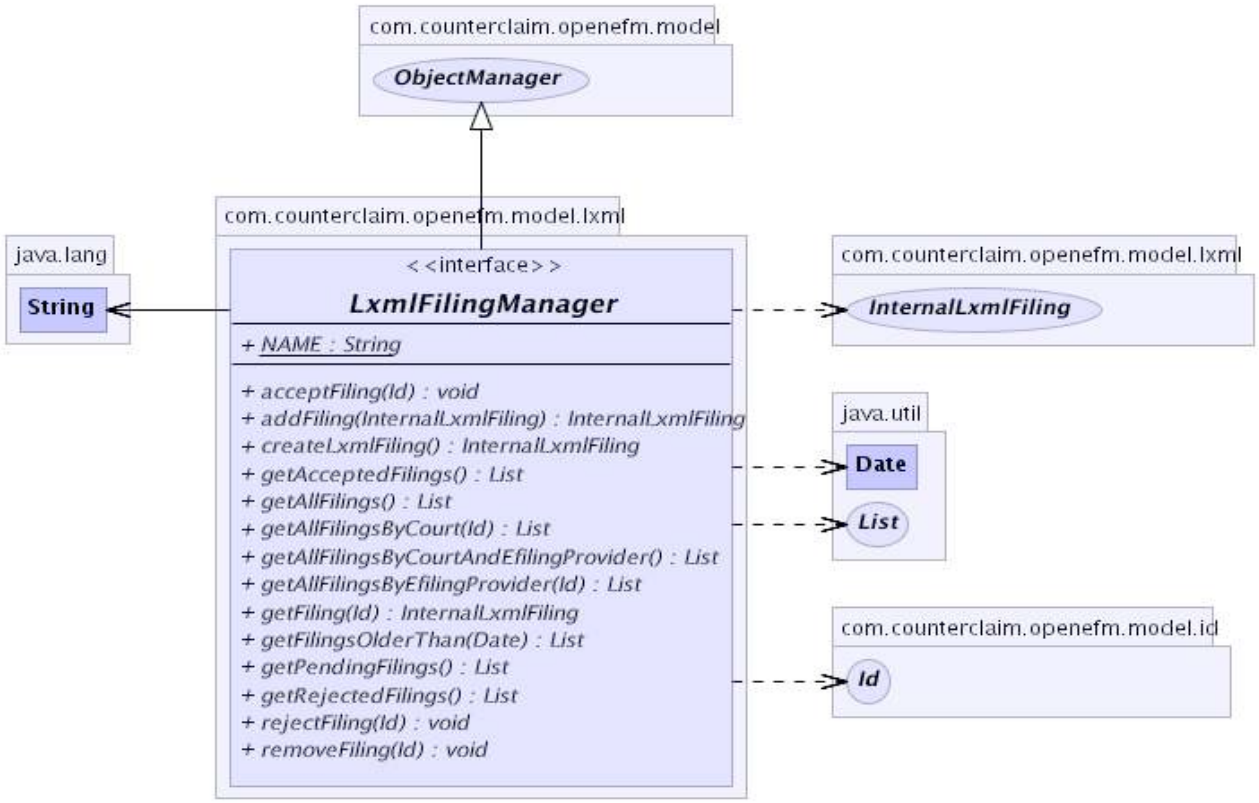
H. LxmlFiling Object



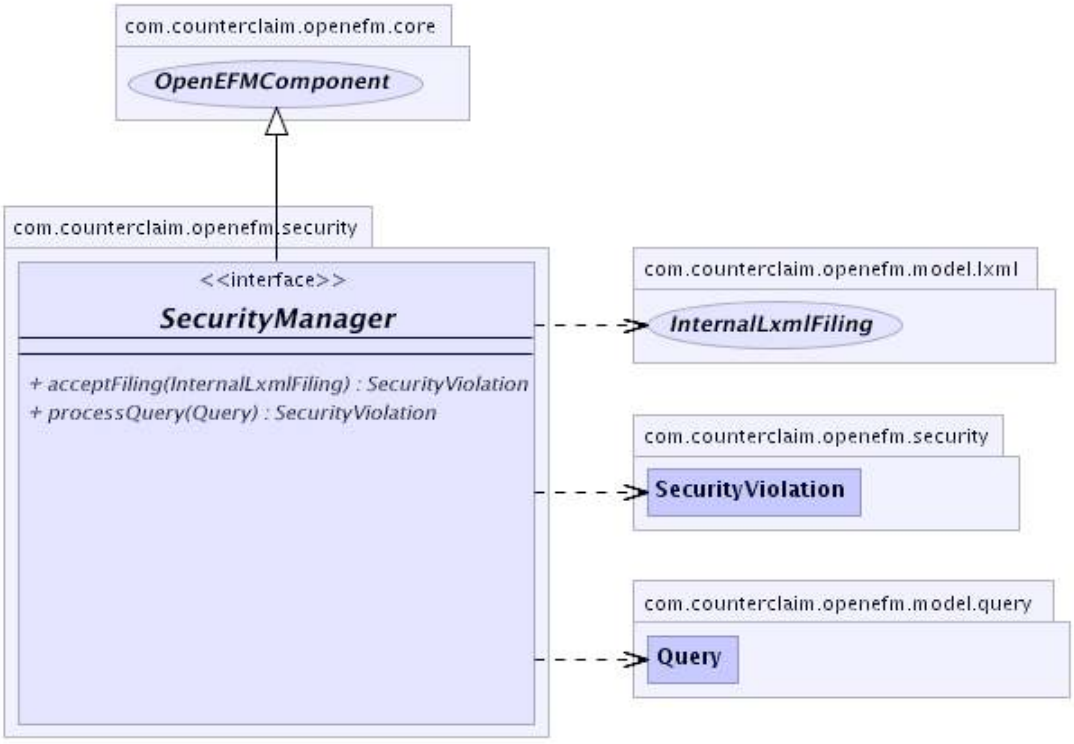
I. InternalLxmlFiling Object



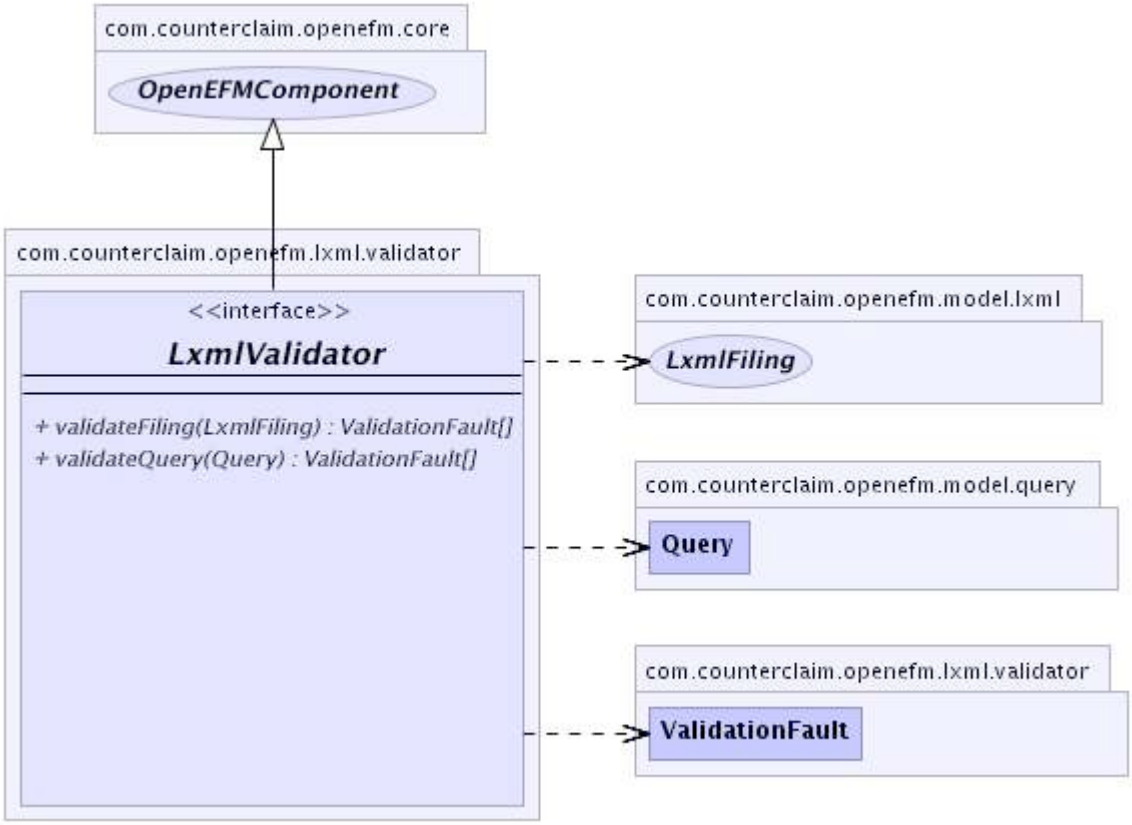
J. LxmlFilingManager Object



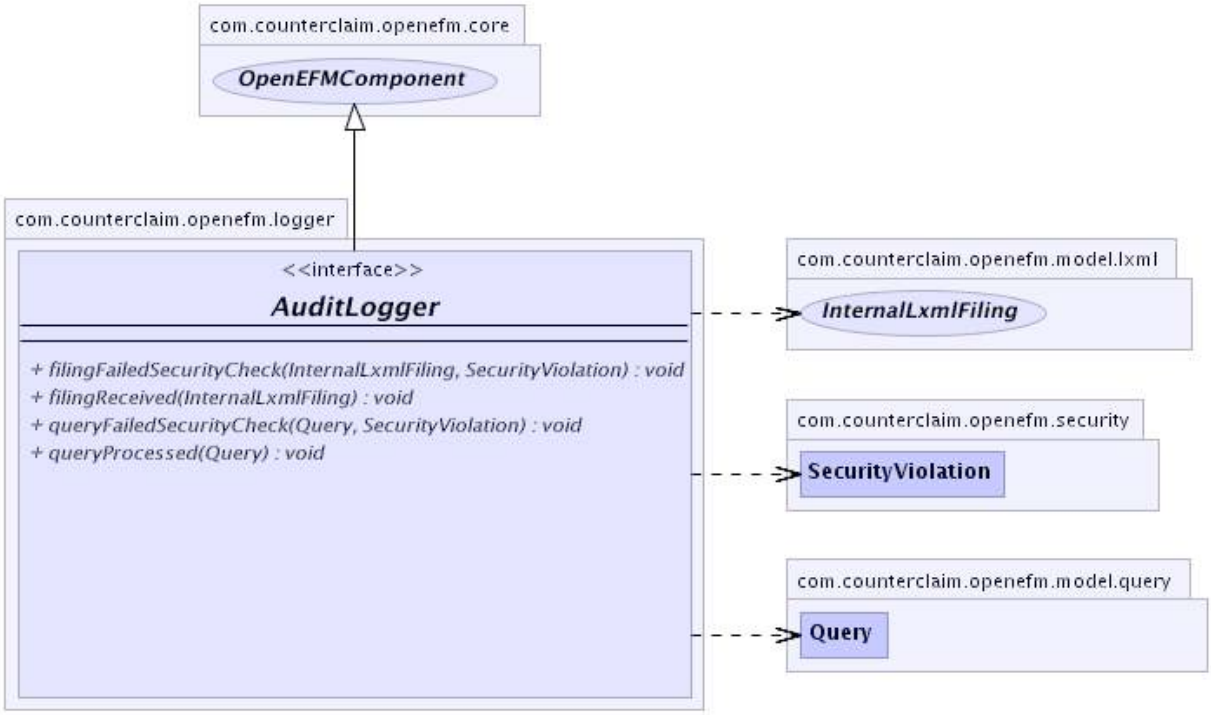
K. SecurityManager Object



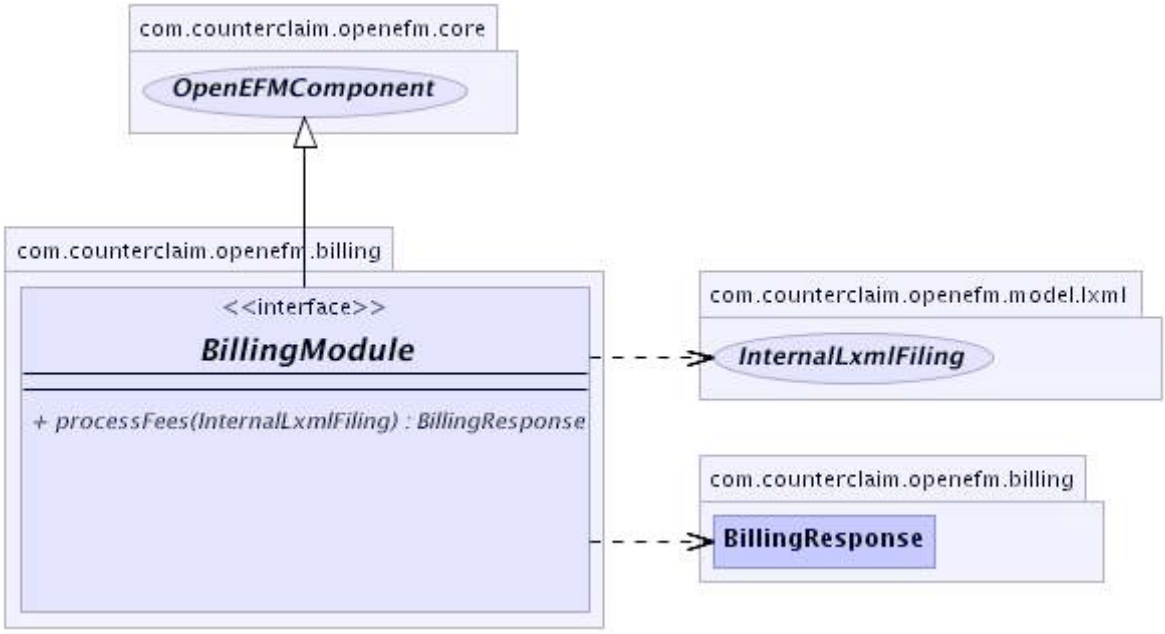
L. LxmlValidator Object



M. AuditLogger Object



N. BillingModule Object



APPENDIX D
REVISION HISTORY

REVISION HISTORY

Version	Date	Revised By	Description
0.1	1/25/04	Mr. Jim Beard counterclaim, Inc.	Initial creation of the document. Generated content for the High-Level and Detailed Design sections of the document.
0.2	1/26/04	Mr. Beard counterclaim, Inc.	The second version of this document introduced content for the Purpose, Scope, and Related Documents subsections and for the Development Requirements section.
0.3	1/29/04	Mr. Beard counterclaim, Inc.	Changed format to comply more closely with MTG Management Consultants, L.L.C.'s standards.
0.4	2/3/04	Mr. Beard counterclaim, Inc.	Updated the design after a design meeting held on January 30, 2004.
0.5	2/6/04	Mr. Beard counterclaim, Inc.	Updated the Development Requirements section. Added content to Graphical User Interface section.
0.6	10/28/04	Mr. Beard counterclaim, Inc.	Updated pending completion of the project.