IBM Cúram Social Program Management

Cúram Evidence Developers Guide

Version 6.0.5

IBM Cúram Social Program Management

Cúram Evidence Developers Guide

Version 6.0.5

Note
Before using this information and the product it supports, read the information in "Notices" on page 35

Revised: May 2013

This edition applies to IBM Cúram Social Program Management v6.0 5 and to all subsequent releases unless otherwise indicated in new editions.

Licensed Materials - Property of IBM.

Contents

Figures

Tables

Chapter 1. Introduction

1.1 Purpose

The purpose of this document is to provide assistance to developers intending to implement evidence solutions using Cúram's Evidence solution. It outlines common pieces of evidence maintenance functionality and describes how a developer can design / implement such functionality.

1.2 Prerequisites

The readers should be familiar with the evidence capturing aspect of case management as well as its use in determining eligibility and entitlement on a case. They should also have read "The Evidence Pattern" in the Cúram Evidence Solutions guide.

1.3 Audience

This document is targeted at a technical audience, both developers and architects, intending to implement evidence solutions using Cúram's Evidence framework.

Chapter 2. Server / Client Evidence Components

2.1 Server Side Artifacts

All of the Evidence server side infrastructure artifacts are shipped in the "curam.core.sl.infrastructure.impl" package. The key elements found here include the Evidence Controller / Evidence Controller Hook (see section 3.8) classes and the Evidence Interfaces. The Interfaces form part of the Interface Hierarchy. The Participant Evidence Interface and Evidence Interface both extend the parent Interface, Standard Evidence Interface. These Evidence Interfaces will be the artifacts of most interest to designers / developers as each evidence entity will need to implement this interface.

2.1.1 Standard Evidence Interface

The Standard Evidence Interface defines the following methods which are common to both inheriting interfaces. The interface and its associated methods are shown below with the appropriate javadoc comments:

```
/*
 * Copyright 2005-2006,2011 Curam Software Ltd.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Curam Software, Ltd. ("Confidential Information").  You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Curam Software.
 */
package curam.core.sl.infrastructure.impl;

import curam.core.sl.infrastructure.entity.struct
  .AttributedDateDetails;
import curam.core.sl.infrastructure.struct.EIEvidenceKey;
import curam.core.sl.infrastructure.struct.EIEvidenceKeyList;
import
  curam.core.sl.infrastructure.struct.EIFieldsForListDisplayDtls;
import curam.core.sl.infrastructure.struct.ValidateMode;
import curam.core.struct.CaseKey;
import curam.util.exception.AppException;
import curam.util.exception.InformationalException;
import curam.util.type.Date;

/**
 * This interface is a key component of the Curam
 * Evidence Solution. Implementations hoping to manage evidence
 * via the Evidence Solution must ensure that the
 * evidence entities contained within the solution implement the
 * Evidence Interface. By doing this, the evidence is utilizing
 * the Evidence Controller pattern whereby a lot of the common
 * business functions for maintaining evidence are contained
 * within the out-of-the-box evidence infrastructure.
 *
 * This interface is the super interface that will be
 * extended by other evidence interfaces that wish to provide
 * custom functionality for that type of evidence. The methods
 * defined on this evidence are common to any interface that
 * extends it.
 */
public interface StandardEvidenceInterface {

  // _____________________________________________________________
  /**
   * Method for calculating case attribution dates. The
   * calculation of evidence attribution is an integral part of a
   * evidence solution as it determines the period of
   * time for which a piece of evidence is effective. The
   * implementation of this function will contain the logic that
   * derives the appropriate effective period for the evidence of
   * a particular type.
   *
   * @param caseKey
   *          Contains a case identifier
   * @param evKey
   *          Contains the evidenceID / evidenceType pairing of
   *          the evidence to be attributed
   *
   * @return Case attribution details
   */
  AttributedDateDetails calcAttributionDatesForCase(
    CaseKey caseKey, EIEvidenceKey evKey)
      throws AppException, InformationalException;

  // _____________________________________________________________
  /**
   * Retrieves a summary of evidence details which are used to
   * populate the 'Details' column on the following evidence
   * pages:
   *
   *  - All evidence workspace pages
   *  - Apply changes page
   *  - Apply user changes page
```

2.1.2 Evidence Interface

The Evidence Interface and its associated methods are shown below with the appropriate javadoc comments:

```
/*
  * Copyright 2005-2007 Curam Software Ltd.
  * All rights reserved.
  *
  * This software is the confidential and proprietary
  * information of Curam  Software, Ltd. ("Confidential
  * Information"). You shall not disclose such Confidential
  * Information and shall use it only in accordance with the
  * terms of the license agreement you entered into with
  * Curam Software.
  */

 package curam.core.sl.infrastructure.impl;

 import curam.core.sl.infrastructure.struct
   .AttributedDateDetails;
 import curam.core.struct.CaseHeaderKey;
 import curam.util.exception.AppException;
 import curam.util.exception.InformationalException;

 /**
  * This interface extends the StandardEvidenceInterface,
  * therefore any class that implements EvidenceInterface
  * must provide its own implementations of the methods
  * defined in the standard interface. Any methods specific
  * to "classic" (i.e. not participant) evidence are to be
  * defined in this interface.
  */
 public interface EvidenceInterface
   extends StandardEvidenceInterface {

  // ___________________________________________________________________
  /**
   * Transfers evidence from one case to another.
   *
   * @param details
   *          Contains the evidenceID / evidenceType pairings of
   *          the evidence to be transferred and the transferred
   * @param fromCaseKey
   *          The case from which the evidence is being
   *          transferred
   * @param toCaseKey
   *          The case to which the evidence is being
   *          transferred
   */
  void transferEvidence(EvidenceTransferDetails details,
    CaseHeaderKey fromCaseKey, CaseHeaderKey toCaseKey)
      throws AppException, InformationalException;

}
```

2.1.3 Participant Evidence Interface

The Participant Evidence Interface and its associated methods are shown below with the appropriate javadoc comments:

```
/*
 * Copyright 2007 Curam Software Ltd.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Curam Software, Ltd. ("Confidential Information").  You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Curam Software.
 */
package curam.core.sl.infrastructure.impl;

import java.util.ArrayList;

import curam.core.sl.infrastructure.struct.EIEvidenceKey;
import curam.core.sl.infrastructure.struct.EIEvidenceKeyList;
import curam.core.sl.struct.ConcernRoleIDKey;
import curam.util.exception.AppException;
import curam.util.exception.InformationalException;

/**
 * This interface extends the StandardEvidenceInterface therefore
 * any class that implements ParticipantEvidenceInterface must
 * provide its own implementations of the methods defined in the
 * standard interface. Any methods specific to participant
 * evidence be defined in this interface.
 */
public interface ParticipantEvidenceInterface
  extends StandardEvidenceInterface {

  // ___________________________________________________________________
  /**
   * Method to check if the attributes that changed during a
   * modify require reassessment to be run when they are applied.
   *
   * @param attributesChanged
   *          - A list of Strings. Each represents the name of an
   *            attribute that changed
   *
   * @return true if Reassessment required
   */
  boolean checkForReassessment(ArrayList attributesChanged)
    throws AppException, InformationalException;

  // ___________________________________________________________________
  /**
   * Method for creating the snapshot record related to a
   * participant evidence record.
   *
   * @param key
   *         Contains an evidenceID / evidenceType pairing
   *
   * @return The uniqueID and the evidence type of the Snapshot
   *         record.
   */
  EIEvidenceKey createSnapshot(EIEvidenceKey key)
    throws AppException, InformationalException;

  // ___________________________________________________________________
  /**
   * Method to compare attributes on two records of the same
   * entity type. It then returns an ArrayList of strings with
   * the names of each attribute that was different between them.
   *
   * @param key
   *          - Contains an evidenceID / evidenceType pairing
   * @param dtls
   *          - a struct of the same type as the key containing
   *            the attributes to be compared against
   *
   * @return A list of Strings. Each represents an attribute name
   *         that differed.
   */
```

Adopting an interface approach enforces a pattern upon entity design/development as each entity must implement the same interface. This approach allows the Cúram Enterprise FrameworkFramework to provide as much common functionality as possible so that custom implementations can concentrate more on business aspects of evidence maintenance, e.g.validations. Each evidence entity must implement the Evidence Interface to have access to the Evidence Controller class.This class implements the common business logic across all evidence entities and the custom business logic specific to each evidence entity.

2.1.4 Accessing Non-modeled Functions

When the Evidence Interfaces are implemented by evidence entities, the methods defined by these interfaces will be implemented by those entities. These methods will of course be non-modeled so will only exist on the evidence entity impl classes. In order to access the non-modeled functions, it's necessary to cast from the impl class. Examples of this can be seen in the entity program listings later in section 3.2 of this document. This casting mechanism will not work though unless the factory class is extending the impl class as opposed to the base class. The only way that this can be achieved, if no non-stereotyped functions are being added to the class, is to add a non-stereotyped dummy function. If this is not done, it will result in a runtime error when the casting is executed.

2.2 Client Side Artifacts

The client side infrastructure artifacts are located inside the..\webclient\components\core\Evidence Infrastructure directory. This folder primarily contains uim and vim client pages. The vim files will typically be included inside solution specific uim pages to manage generic evidence details whereas the uim pages contain complete out-of-the-box evidence maintenance functionality.

The key benefit of the.im files is that they can be changed in line with any enhancements made to the evidence maintenance solution without any impact on specific implementations, i.e. the upgrade is seamless.

Examples of infrastructural.vim files are as follows:

- Evidence_createHeader.vim
- Evidence_modifyHeader.vim
- Evidence_viewHeader.vim
- Evidence_viewHeaderForModal.vim

These artifacts manage the infrastructural attributes of evidence maintenance and should be included in create, modify and view evidence pages. This will be highlighted later when a sample implementation of the Evidence solution is discussed. Some further examples of vim files include:

- Evidence_typeWorkspace.vim
- Evidence_workspaceInEditHighLevelView.vim
- Evidence_workspaceActiveHighLevelView.vim

These artifacts are used to populate evidence workspaces. An evidence workspace is a central location for managing evidence. The above vim files will be included by workspace.uim pages.

Some examples of infrastructural uim pages which provide entire evidence maintenance functions are:

- Evidence_applyChanges1.uim
- Evidence_addNewEvidence.uim
- Evidence_dashboard.uim

Evidence_applyChanges1 lists all work-in-progress evidence, i.e. all new and updated evidence or evidence that is pending removal. The display and action bean on this page live on the Evidence facade which is part of the centralized evidence maintenance functionality.

Evidence_addNewEvidence lists all possible evidence types, filtered by category, and launches an appropriate create page for each.

Evidence_dashboard lists all evidence types on the given case broken into categories. It highlights which types have In Edit evidence recorded and which have verifications or issues outstanding.

Note: It is important to note that in some cases.vim files found in the client infrastructure package are actually included in infrastructure pages. For instance, Evidence_dashboardView.vim is included inside the Evidence_dashboard page and Evidence_flowView.vim is included inside the Evidence_flow page.

Chapter 3. Developing an Evidence Solution

3.1 Administration

3.1.1 Evidence Metadata

The Evidence Metadata entity contains metadata information relating to each evidence type. This entity must be populated before evidence maintenance can proceed. A number of evidence page names, including the view and modify page names, are included in the metadata. These page names are retrieved at runtime via evidence infrastructure resolve scripts and via implementations of the Evidence Type interface on the server. The records on the Evidence Metadata entity are effective dated to facilitate pages changing over time, due to legislation for example.

3.1.2 Product Evidence Link

The Product Evidence Link entity links evidence to a product. In some circumstances, evidence may be stored at the Integrated Case level but only some of this evidence may apply to a given product on the Integrated Case. To know which evidence should be attributed to a given product, a lookup of this entity is performed as part of the attribution processing and only evidence linked to the product is attributed.

3.2 Common Evidence Maintenance Operations

In this section, some common evidence maintenance operations are outlined. This is done using sequence diagrams, client screenshots and server code snippets from the a sample product implementation. This product is used for demonstration purposes only.

3.2.1 Create Evidence

The development, both client and server, of a create evidence operation is outlined here.

3.2.1.1 Create Evidence Sequence Diagram

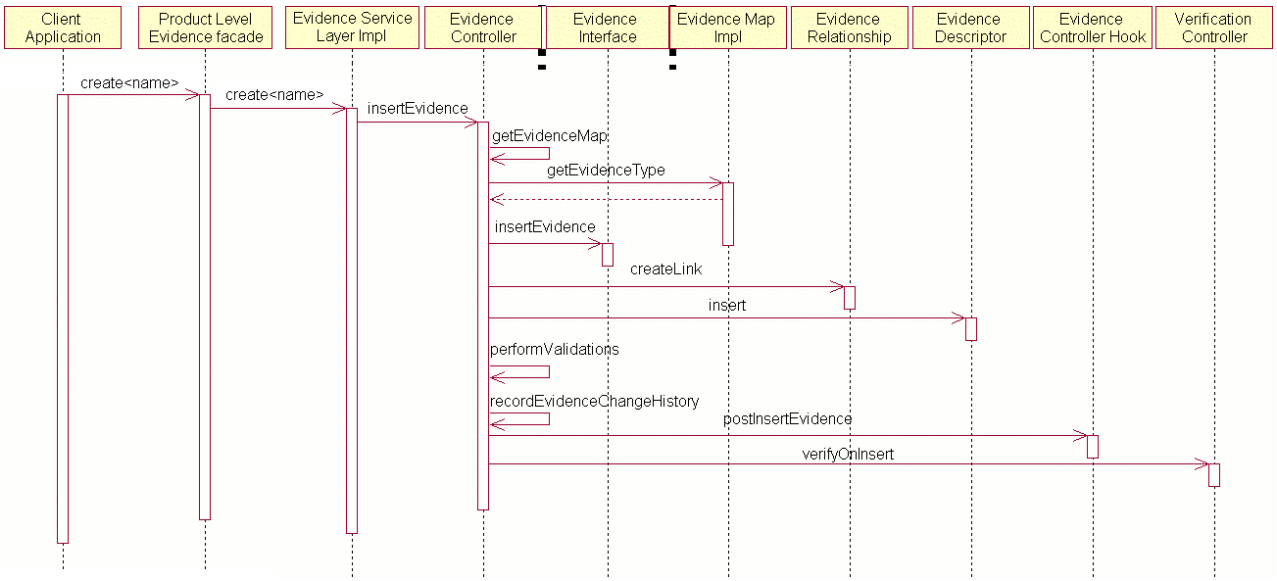

Figure 1. Sequence Diagram for Creating Evidence

3.2.1.2 Client - Screen to Be Developed

The client page to be developed must include the evidence infrastructure page Evidence_createHeader.vim. This included.vim page facilitates the management of infrastructure attributes. For example, the Evidence Descriptor's receivedDate attribute is currently managed through this infrastructure page. If, at some point in the future, additional attributes which need to be managed through the create function were added to the Evidence Descriptor entity, then these attributes could be mapped through this infrastructure page. Hence, this requires just a once-off infrastructure change rather than many changes to custom artifacts.

3.2.1.3 Server - Methods to Be Implemented

- *SEGEvidenceMaintainenance.createAssetEvidence facade operation*

  The facade operation calls the evidence service layer implementation.

```
// _____________________________________________________
/**
 * Creates an Asset evidence record.
 *
 * @param dtls Details of the new evidence record to be created.
 *
 * @return The details of the created record.
 */
 public ReturnEvidenceDetails createAssetEvidence(
  AssetEvidenceDetails dtls)
  throws AppException, InformationalException {

  // set the informational manager for the transaction
  TransactionInfo.setInformationalManager();

  // Asset evidence manipulation object
  Asset evidenceObj = AssetFactory.newInstance();

  // return object
  ReturnEvidenceDetails createdEvidenceDetails =
    new ReturnEvidenceDetails();

  // create the Asset record and populate the return details
  createdEvidenceDetails =
    evidenceObj.createAssetEvidence(dtls);

  createdEvidenceDetails.warnings =
    EvidenceControllerFactory.newInstance().getWarnings();

  return createdEvidenceDetails;
}
```

- *Asset.createAssetEvidence* service layer operations

  These overloaded service layer operations call the Evidence Controller infrastructure function for inserting evidence.

```
// _____________________________________________________
/**
 * Creates a Asset record.
 *
 * @param dtls Contains Asset evidence record creation details.
 *
 * @return the new evidence ID and warnings.
 */
public ReturnEvidenceDetails createAssetEvidence(
  AssetEvidenceDetails dtls)
  throws AppException,InformationalException {

  return createAssetEvidence(dtls, null, null, false);
}
```

```
// ____________________________________________________________
/**
 * Creates a Asset record.
 *
 * @param dtls Contains Asset evidence record creation details.
 *
 * @param sourceEvidenceDescriptorDtls If this function is called
 * during evidence sharing, this parameter will be non-null and
 * it represents the header of the evidence record being shared
 * (i.e. the source evidence record)
 *
 * @param targetCase If this function is called during evidence
 * sharing, this parameter will be non-null and it represents the
 * case the evidence is being shared with.
 *
 * @param sharingInd A flag to determine if the function is
 * called in evidence sharing mode. If false, the function is
 * being called as part of a regular create.
 *
 * @return the new evidence ID and warnings.
 */
public ReturnEvidenceDetails createAssetEvidence(
  AssetEvidenceDetails dtls,
  EvidenceDescriptorDtls sourceEvidenceDescriptorDtls,
  CaseHeaderDtls targetCase, boolean sharingInd)
  throws AppException,InformationalException {

  // validate the mandatory fields
  validateMandatoryDetails(dtls);

  EvidenceControllerInterface evidenceControllerObj =
    (EvidenceControllerInterface)
      EvidenceControllerFactory.newInstance();
  EvidenceDescriptorInsertDtls evidenceDescriptorInsertDtls =
    new EvidenceDescriptorInsertDtls();

  ReturnEvidenceDetails createdEvidence =
    new ReturnEvidenceDetails();

  if (sharingInd) {

    EvidenceDescriptorDtls sharedDescriptorDtls =
      evidenceControllerObj.shareEvidence(
        sourceEvidenceDescriptorDtls,
        targetCase);

    // Return the evidence ID and warnings
    createdEvidence.evidenceKey.evidenceID =
      sharedDescriptorDtls.relatedID;
    createdEvidence.evidenceKey.evType =
      sharedDescriptorDtls.evidenceType;

  } else {

    // As there is no participant associated with this evidence
    // we must retrieve the case participant to set the evidence
    // descriptor participant.
    CaseHeaderKey caseHeaderKey = new CaseHeaderKey();
    caseHeaderKey.caseID = dtls.caseIDKey.caseID;
    evidenceDescriptorInsertDtls.participantID =
      CaseHeaderFactory.newInstance().readCaseParticipantDetails(
        caseHeaderKey).concernRoleID;

    // Evidence descriptor details
    evidenceDescriptorInsertDtls.caseID = dtls.caseIDKey.caseID;
    evidenceDescriptorInsertDtls.evidenceType =
      CASEEVIDENCE.ASSET;
```

```
    evidenceDescriptorInsertDtls.receivedDate =
      dtls.descriptor.receivedDate;

    // Upon creation, the change reason should be Initial
    evidenceDescriptorInsertDtls.changeReason =
      EVIDENCECHANGEREASON.INITIAL;

    // Evidence Interface details
    EIEvidenceInsertDtls eiEvidenceInsertDtls =
      new EIEvidenceInsertDtls();
    eiEvidenceInsertDtls.descriptor.assign(
      evidenceDescriptorInsertDtls);
    eiEvidenceInsertDtls.evidenceObject = dtls.dtls;

    // Insert the evidence
    EIEvidenceKey eiEvidenceKey =
      evidenceControllerObj.insertEvidence(eiEvidenceInsertDtls);

    // Return the evidence ID and warnings
    createdEvidence.evidenceKey.evidenceID =
      eiEvidenceKey.evidenceID;
    createdEvidence.evidenceKey.evType =
      eiEvidenceKey.evidenceType;
    createdEvidence.warnings =
      evidenceControllerObj.getWarnings();
  }

  return createdEvidence;
}
```

3.2.2 Modify Evidence

The development, both client and server, of a modify evidence operation is outlined here.

3.2.2.1 Modify Evidence Sequence Diagram

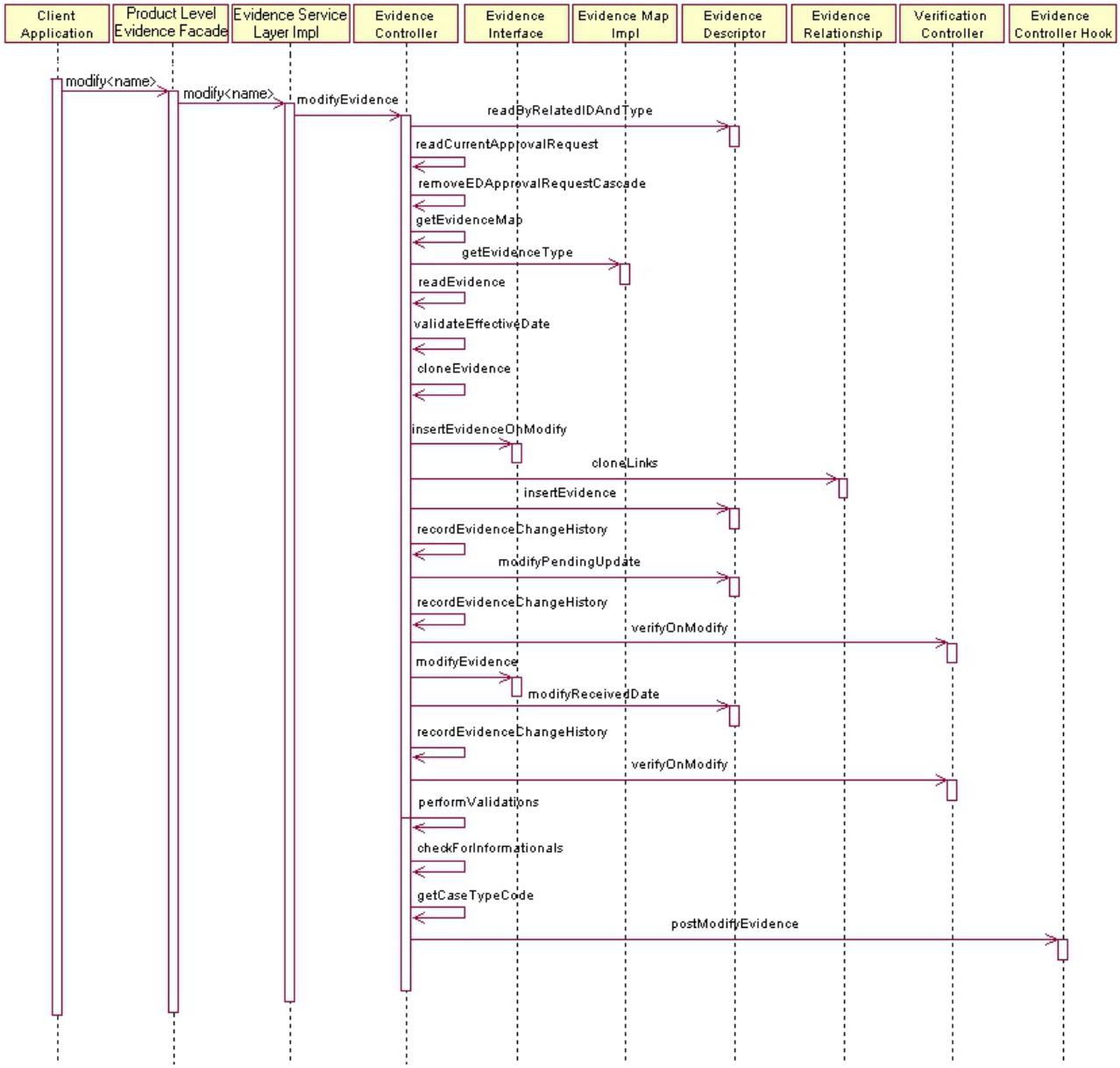

Figure 2. Sequence Diagram for Modifying Evidence

3.2.2.2 Client - Screen to Be Developed

The client page to be developed must include the evidence infrastructure page Evidence_modifyHeader1.vim. This included.vim page facilitates the viewing / modification of some infrastructure attributes. For example, received date can be viewed or modified via this.vim. Also, change reason and effective date of change can be set on the edited record. If, at some point in the future, additional attributes which need to be managed through the modify function were added to the Evidence Descriptor entity, then these attributes could be mapped through this infrastructure page. Hence, this requires just a once-off infrastructure change rather than many changes to custom artifacts.

The inclusion of Evidence_modifyHeader1.vim facilitates the following three types of evidence modification:

- Editing Evidence In Place

This refers to the modification of incorrect data on a piece of evidence which has not yet been activated. In this scenario, if the effective date is modified an error will be thrown informing the user that the date can only be modified when updating an active record.

- Evidence Correction

  An evidence correction occurs when a piece of data on an active evidence record is modified resulting in the current active record being superseded. In this scenario, the effective date field must not be modified as this will result in a new record in the succession being created - see evidence succession.

- Evidence Succession

  If the user modifies the effective date when updating a piece of active evidence, they are specifying a new record in the succession set, i.e. the new record will have the same successionID as the active record. Therefore, the active record will essentially be copied and made effective from the effective date specified by the user and the update applied to this record.

  Note: Activation of newly created records in a succession will cause reattribution of records in that succession set.

3.2.2.3 Server - Methods to Be Implemented

- *SEGEvidenceMaintenance.modifyAssetEvidence facade operation*

  The facade operation calls the evidence service layer implementation.

```
// _____________________________________________________________
/**
 * Modifies an Asset evidence record.
 *
 * @param details The modified evidence details.
 *
 * @return The details of the modified evidence record.
 */
public ReturnEvidenceDetails modifyAssetEvidence(
  AssetEvidenceDetails dtls)
    throws AppException, InformationalException {

  // set the informational manager for the transaction
  TransactionInfo.setInformationalManager();

  // Asset evidence manipulation object
  Asset evidenceObj = AssetFactory.newInstance();

  // return object
  ReturnEvidenceDetails modifiedEvidenceDetails =
    new ReturnEvidenceDetails();

  // modify the Asset record and populate the return details
  modifiedEvidenceDetails =
    evidenceObj.modifyAssetEvidence(dtls);

  modifiedEvidenceDetails.warnings =
    EvidenceControllerFactory.newInstance().getWarnings();

  return modifiedEvidenceDetails;
}
```

- *Asset.modifyAssetEvidence* service layer operation

  This service layer operation calls the Evidence Controller infrastructure function for modifying evidence.

```
// _____________________________________________________________
/**
 * Modifies an Asset record.
 *
 * @param dtls Contains Asset evidence record modification
 *             details.
 *
 * @return The modified evidence ID and warnings.
```

```
 */
public ReturnEvidenceDetails modifyAssetEvidence
  (AssetEvidenceDetails details)
    throws AppException, InformationalException {

  // validate the mandatory fields
  validateMandatoryDetails(details);

  // EvidenceController business object
  EvidenceControllerInterface evidenceControllerObj =
    (EvidenceControllerInterface)
      EvidenceControllerFactory.newInstance();

  EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();

  //
  // Call the EvidenceController to modify the evidence
  //

  eiEvidenceKey.evidenceID = details.dtls.evidenceID;
  eiEvidenceKey.evidenceType = CASEEVIDENCE.ASSET;

  // Create the evidence interface modification struct and assign
  // the details
  EIEvidenceModifyDtls eiEvidenceModifyDtls =
    new EIEvidenceModifyDtls();
  eiEvidenceModifyDtls.descriptor.receivedDate =
    details.descriptor.receivedDate;
  eiEvidenceModifyDtls.descriptor.versionNo =
    details.descriptor.versionNo;
  eiEvidenceModifyDtls.descriptor.effectiveFrom =
    details.descriptor.effectiveFrom;
  eiEvidenceModifyDtls.descriptor.changeReceivedDate =
    details.descriptor.changeReceivedDate;
  eiEvidenceModifyDtls.descriptor.changeReason =
    details.descriptor.changeReason;
  eiEvidenceModifyDtls.evidenceObject = details.dtls;

  evidenceControllerObj.modifyEvidence(
    eiEvidenceKey, eiEvidenceModifyDtls);

  //
  // Return details from the modify operation
  //

  ReturnEvidenceDetails returnEvidenceDetails =
    new ReturnEvidenceDetails();
  returnEvidenceDetails.evidenceKey.evidenceID =
    eiEvidenceKey.evidenceID;
  returnEvidenceDetails.evidenceKey.evType =
    eiEvidenceKey.evidenceType;
  returnEvidenceDetails.warnings =
    evidenceControllerObj.getWarnings();


  return returnEvidenceDetails;
}
```

3.2.3 Read Evidence

The development, both client and server, of a read evidence operation is outlined here.

3.2.3.1 View Evidence Sequence Diagram

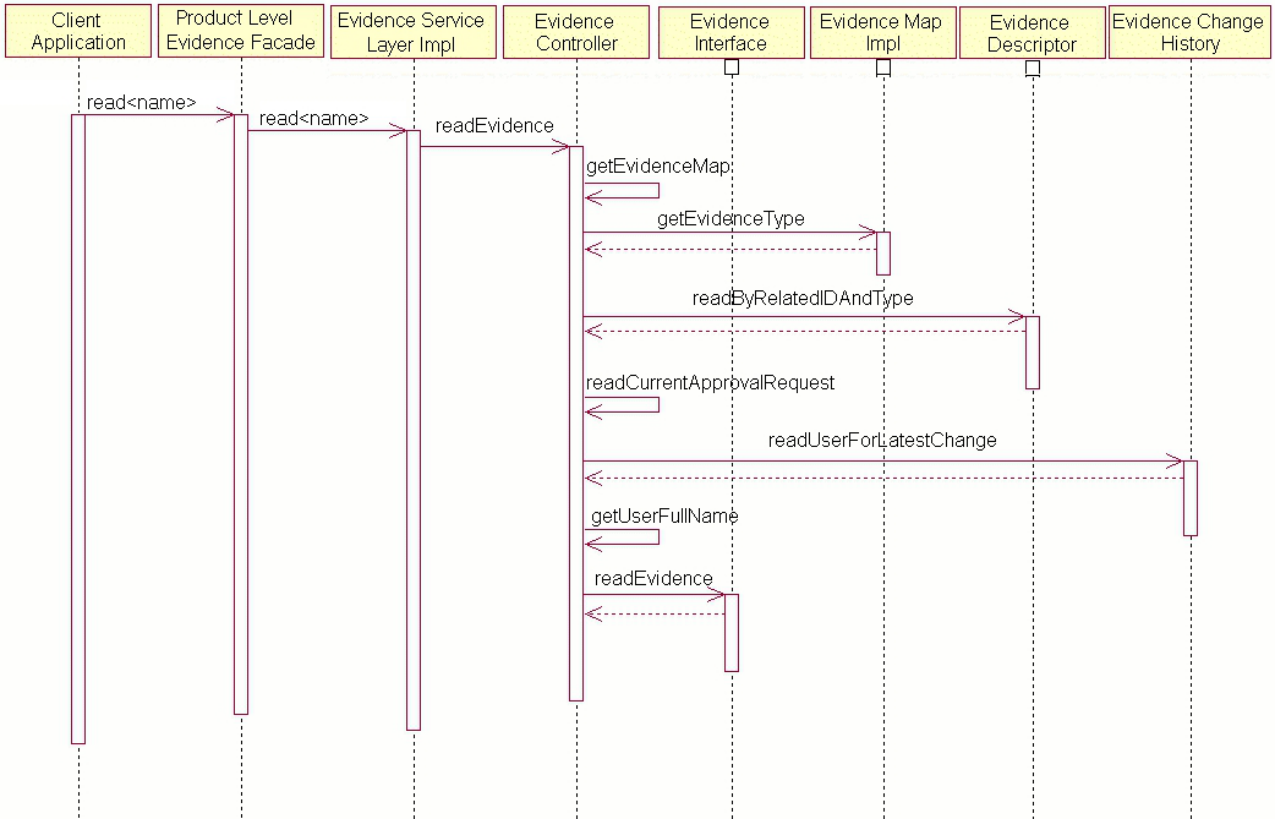

Figure 3. Sequence Diagram for Viewing Evidence

3.2.3.2 Client - Screen to Be Developed

The client page includes the evidence infrastructure page Evidence_viewHeaderForModal.vim. This included.vim facilitates the viewing of some infrastructure attributes.

3.2.3.3 Server - Methods to Be Implemented

- *SEGEvidenceMaintenance.readAssetEvidence facade operation*

  The facade operation calls the evidence service layer implementation.

```
// _____________________________________________________________
/**
 * Reads an Asset evidence record.
 *
 * @param key Identifies the evidence record to read.
 *
 * @return The details of the evidence record.
 */
public ReadAssetEvidenceDetails readAssetEvidence(
  EvidenceCaseKey key)
    throws AppException, InformationalException {

  // Asset evidence manipulation object
  Asset evidenceObj = AssetFactory.newInstance();

  // return object
  ReadAssetEvidenceDetails readEvidenceDetails =
    new ReadAssetEvidenceDetails();

  // read the Asset record and populate the return details
  readEvidenceDetails = evidenceObj.readAssetEvidence(key);
```

```
  return readEvidenceDetails;

}
```

This service layer operation calls the Evidence Controller infrastructure function for reading evidence.

```
// ____________________________________________________________
/**
 * Reads an Asset record.
 *
 * @param key contains ID of record to read.
 *
 * @return Asset evidence details read.
 */
public ReadAssetEvidenceDetails readAssetEvidence(
  EvidenceCaseKey key)
    throws AppException, InformationalException {

  // EvidenceController business object
  EvidenceControllerInterface evidenceControllerObj =
    (EvidenceControllerInterface)
      EvidenceControllerFactory.newInstance();

  EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
  eiEvidenceKey.evidenceID = key.evidenceKey.evidenceID;
  eiEvidenceKey.evidenceType = CASEEVIDENCE.ASSET;

  // Retrieve the evidence details
  EIEvidenceReadDtls eiEvidenceReadDtls =
    evidenceControllerObj.readEvidence(eiEvidenceKey);

  // Retrieve the evidence descriptor details
  EvidenceDescriptor evidenceDescriptorObj =
    EvidenceDescriptorFactory.newInstance();

  EvidenceDescriptorKey evidenceDescriptorKey =
    new EvidenceDescriptorKey();
  evidenceDescriptorKey.evidenceDescriptorID =
    eiEvidenceReadDtls.descriptor.evidenceDescriptorID;

  EvidenceDescriptorDtls evidenceDescriptorDtls =
    evidenceDescriptorObj.read(evidenceDescriptorKey);

  //
  // Return the evidence
  //

  ReadAssetEvidenceDetails readEvidenceDetails =
    new ReadAssetEvidenceDetails();
  readEvidenceDetails.descriptor
    .assign(evidenceDescriptorDtls);

  readEvidenceDetails.descriptor.approvalRequestStatus =
    eiEvidenceReadDtls.descriptor.approvalRequestStatus;
  readEvidenceDetails.descriptor.updatedBy =
    eiEvidenceReadDtls.descriptor.updatedBy;
  readEvidenceDetails.descriptor.updatedDateTime =
    eiEvidenceReadDtls.descriptor.updatedDateTime;

  // assign the evidence to the return object
  readEvidenceDetails.dtls.assign(
    (AssetDtls)(eiEvidenceReadDtls.evidenceObject));

  return readEvidenceDetails;
}
```

3.2.4 List Evidence

The development, both client and server, of a list evidence operation is outlined here. The list operation is used to populate an evidence workspace page.

3.2.4.1 List Evidence Sequence Diagram

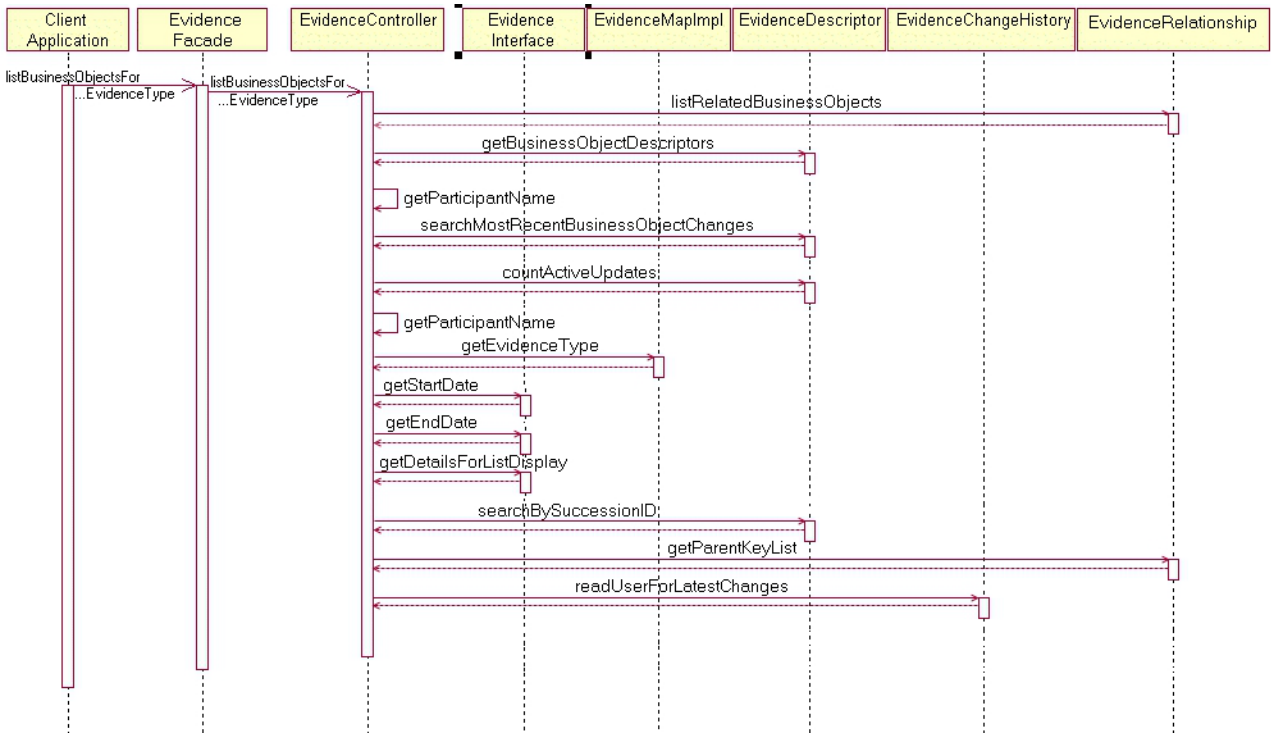

Figure 4. Sequence Diagram for Listing Evidence

3.2.4.2 Server - Methods to Be Developed

Much of the data displayed on the workspace page is retrieved via the Evidence Descriptor entity. The description and period are retrieved via Evidence Interface methods which must be implemented for each evidence type.

- *Asset.getDetailsForListDisplay entity operation*

  The description, or summary details, is retrieved via the getDetailsForListDisplay Evidence Interface method which is implemented by the evidence entities. The implementation of the getDetailsForListDisplay method for the Asset is shown below. This interface function is also used to retrieve summary data when applying, approving, rejecting evidence as well as in evidence sharing, verifications and issues screens.

```
// _____________________________________________________________
/**
 * Gets evidence details for the list display
 *
 * @param key Evidence key containing the evidenceID and
 * evidenceType
 *
 * @return Evidence details to be displayed on the list page
 */
public EIFieldsForListDisplayDtls getDetailsForListDisplay(
  EIEvidenceKey key)
  throws AppException, InformationalException {

  // Return object
  EIFieldsForListDisplayDtls eiFieldsForListDisplayDtls =
```

```
      new EIFieldsForListDisplayDtls();

    // Asset entity key
    final AssetKey assetKey = new AssetKey();
    assetKey.evidenceID = key.evidenceID;

    // Read the Asset entity to get display details
    final AssetDtls assetDtls =
      AssetFactory.newInstance().read(assetKey);

    // Set the start / end dates
    eiFieldsForListDisplayDtls.startDate = assetDtls.startDate;
    eiFieldsForListDisplayDtls.endDate = assetDtls.endDate;

    LocalisableString summary = new LocalisableString(
      BIZOBJDESCRIPTIONS.BIZ_OBJ_DESC_ASSET);

    summary.arg(
      CodeTable.getOneItem(SAMPLEASSETTYPE.TABLENAME,
        assetDtls.assetType));

    // Format the amount for display
    TabDetailFormatter formatterObj =
      TabDetailFormatterFactory.newInstance();
    AmountDetail amount = new AmountDetail();
    amount.amount = assetDtls.value;
    summary.arg(formatterObj.formatCurrencyAmount(amount).amount);

    eiFieldsForListDisplayDtls.summary =
      summary.toClientFormattedText();

    return eiFieldsForListDisplayDtls;
  }
```

3.3 Evidence Dashboard and EvidenceFlow

The Evidence Dashboard and EvidenceFlow are user interface constructs introduced to assist user navigation to all evidence on a case. No custom code is required in order to configure these for a custom case as these are infrastructural.

From these pages, a user can select a particular evidence type which should open the respective evidence workspace for that type of evidence. In the case of the Dashboard, this will open in a new tab, whereas the EvidenceFlow will redirect the bottom portion of the page.

The existence of 'In Edit' evidence records, outstanding verifications and outstanding issues are all highlighted graphically.

The list of evidence types on the case may be split into categories on these pages, by defining the category on the AdminICEvidenceLink table for Integrated Cases, or on the ProductEvidenceLink table for Product Deliveries.

3.4 Validations

The infrastructure facilitates the validation of work-in-progress changes. The validate page can be used either at a case level or on an individual evidence type.

The purpose of the case level validate page is to provide a means to test validations in advance of applying the changes. For some products, the full evidence set may be quite sizeable resulting in the apply changes listing containing a considerable number of evidence changes of varying evidence types. In that scenario, the individual evidence type validate page may make it easier to associate a validation

message with the correct evidence record. The validate page allows a user to pre-test the evidence changes. The user can see which validations will fail and fix them before applying the changes.

3.5 More On Validations

Two of the Evidence Interface functions which form part of the infrastructure support for evidence validation are selectForValidations and validate.

The selectForValidations function will typically be used to select all evidences which are related to or are dependant on the piece of evidence being validated. An example of this would be the modification of an amount on a parent evidence record. As part of the validation of the parent evidence, a check might need to be performed to ensure the sum of the child evidence records does not exceed the modified parent amount.

When a user applies changes to evidence records, the Evidence Controller calls out to the selectForValidations interface function on the entities for each evidence record. The logic within this method retrieves all related 'Active' and 'In Edit' evidences within the hierarchy for validation. For instance, if we are validating a child evidence record within a parent-child-grandchild relationship structure, both parent evidence and grandchild evidence are retrieved for the validation processing.

Once processing returns to the Evidence Controller, a filter is applied to the list of evidence. This filters the input list and leaves only 'Active' records, or 'In Edit' records as appropriate depending on whether the function must validate against work-in-progress or active only evidence. This filtered list is then passed to the validate function where custom validation is applied.

The program listing below shows a selectForValidations implementation used in the Asset demo.

```
// _____________________________________________________
  /**
   * Selects all the records for validations
   *
   * @param evKey Contains an evidenceID / evidenceType pairing
   *
   * @return List of evidenceID / evidenceType pairings
   */
  public EIEvidenceKeyList selectForValidation(
    EIEvidenceKey evKey)
    throws AppException, InformationalException {

    // Return object
    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();

    // Casting to impl due to calling non-modeled interface
    curam.seg.evidence.entity.intf.AssetOwnership
      assetOwnershipObj =
        (curam.seg.evidence.entity.impl.AssetOwnership)
           AssetOwnershipFactory.newInstance();

    eiEvidenceKey.evidenceID = evKey.evidenceID;
    eiEvidenceKey.evidenceType =
      CASEEVIDENCE.ASSET;

    EIEvidenceKeyList eiEvidenceKeyList =
      assetOwnershipObj.readAllByParentID(eiEvidenceKey);

    eiEvidenceKeyList.dtls.add(0, evKey);

    return eiEvidenceKeyList;
  }
```

The code here, on the Asset parent entity, makes a call to the readAllByParentID interface method implementation on the child entity, Asset Ownership. The implementation of the readAllByParentID function on the Asset Ownership is shown in the program listing below.

```
// _____________________________________________________________
/**
 * Read all Asset Ownership records
 *
 * @param key Contains the evidenceID and evidenceType
 *
 * @return A list of evidenceID and evidenceType pairs
 */
public EIEvidenceKeyList readAllByParentID(EIEvidenceKey key)
  throws AppException, InformationalException {

  // Return object
  EIEvidenceKeyList eiEvidenceKeyList = new EIEvidenceKeyList();

  // Create the link entity object
  EvidenceRelationship evidenceRelationshipObj =
    EvidenceRelationshipFactory.newInstance();

  // parent entity key
  ParentKey parentKey = new ParentKey();
  parentKey.parentID = key.evidenceID;
  parentKey.parentType = key.evidenceType;

  // Reads all relationship details for the specified parent
  ChildKeyList childKeyList =
    evidenceRelationshipObj.searchByParent(parentKey);

  // Iterate through the link details list
  for (int i = 0; i < childKeyList.dtls.size(); i++) {

    if (childKeyList.dtls.item(i).childType.equals(
      CASEEVIDENCE.ASSETOWNERSHIP)) {

      EIEvidenceKey listEvidenceKey = new EIEvidenceKey();

      listEvidenceKey.evidenceID =
        childKeyList.dtls.item(i).childID;
      listEvidenceKey.evidenceType =
        childKeyList.dtls.item(i).childType;

      eiEvidenceKeyList.dtls.addRef(listEvidenceKey);
    }
  }

  return eiEvidenceKeyList;

}
```

The function above retrieves all child evidence keys for the specified parent. The childID and childType pairings are returned to the calling mechanism.

3.6 Evidence Attribution

Evidence attribution refers to the assignment of a period of time to a given piece of evidence during which that piece of evidence will be used for entitlement calculations. The attribution period may range from a basic one to one mapping from the business start and end dates through to a more sophisticated algorithm considering any number of factors. This custom logic calculates the attribution period and the evidence controller takes care of synchronizing these with the specified effective dates – see example(s) below. It should also be noted that the attribution from and to dates can be null in which case the piece of evidence is assumed effective from the case start date to the expected end date.

One of the Evidence Interface functions is calcAttributionDatesForCase and the implementation of this function on an entity class is where the attribution from and to dates are determined for evidence on that entity.

3.6.1 Re-attribution

When evidence is modified as part of a succession and subsequently activated, re-attribution of the evidence records in the succession set occurs. A basic example of how this works is shown below:

Business Start Date: 3rd May 2006 (=attribution from date)

Business End Date: 30th July 2006 (=attribution to date)

A succession record is created effective from 5th June 2006. On activation of this record, the evidence is re-attributed and the following attribution records created:

3rd May 2006 to 4th June 2006

5th June 2006 to 30th July 2006

Re-attribution also occurs if evidence in a succession set is removed. For example, if the following three attribution records exist for records in the same succession set

3rd May 2006 to 4th June 2006

5th June 2006 to 30th July 2006

31st July 2006 to 29th Sept 2006

and the evidence record associated with the middle one is removed, applying changes will cause the following re-attribution

3rd May 2006 to 30th July 2006

31st July 2006 to 29th Sept 2006

The attribution record from 5th June 2006 to 30th July 2006 remains on the database but won't be picked up by eligibility processing as the associated evidence is removed, i.e. has a status of 'Canceled'.

3.7 Evidence Relationship

By default, the Evidence infrastructure facilitates the linking of parent-child evidence via the EvidenceRelationship link entity. The structure of the EvidenceRelationship link entity is as follows:

Table 1. Evidence Relationship Link Entity

Evidence Relationship
evidenceRelationshipID
parentID
parentType
childID
childType

This supports the relationship between any parent-child evidence and does away with the necessity for customers to model their own link entities for managing such relationships. When evidence is being

inserted, the generic EvidenceController.insertEvidence function makes a call to the business process EvidenceRelationship.createLink. If a parent type has been specified, i.e. passed in from the client as part of the insert, then a record will be written to the EvidenceRelationship entity linking the child evidence to its parent. Also, a call is made to the business process EvidenceRelationship.cloneLinks directly after the call to the interface operation insertEvidenceOnModify. From cloneLinks, two further calls are made to cloneLinksForParent and cloneLinksForChild.

If customers are using their own link entities to manage relationships, they will need to override the Evidence Relationship business processes for creating and cloning links. The evidence type is available in the input keys of both these functions which means that responsibility can be delegated to the appropriate custom relationship processing based on the evidence type in the key.

3.8 Registering Evidence Implementations

The evidence maintenance pattern requires the set of evidence entities to be registered before they can be used. This is so that the controller can access these evidence entities at runtime.

The Core Cúram Framework does not know in advance which evidence entities will be used for the given evidence maintenance facility associated with a particular product implementation. The evidence types and their implementation must be paired at runtime.

3.8.1 Evidence Registrar Module

Google Guice dependency injection should be used in order to register the different evidence types and their implementations. This can be done by writing a new module class, or adding to a pre existing one. Once this is added to the ModuleCalssName table, then at runtime it will be loaded and the evidence types registered.

Example

```
/*
 * Copyright 2011 Cúram Software Ltd.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Cúram Software, Ltd. ("Confidential Information").  You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Cúram Software.
 */

package curam.seg.evidence.service.impl;


import curam.codetable.CASEEVIDENCE;
import com.google.inject.AbstractModule;
import curam.core.impl.FactoryMethodHelper;
import java.lang.reflect.Method;
import com.google.inject.multibindings.MapBinder;
import curam.core.impl.RegistrarImpl;
import curam.core.impl.Registrar.RegistrarType;


/**
 * A module class which provides registration for all of the
 * evidence hook implementations.
 */
public class SEGRegistrarModule extends AbstractModule {

  @Override
  public void configure() {

    // Register all hook implementations which implement the
```

```
    // interface EvidenceInterface.
    MapBinder<String, Method> evidenceInterfaceMapBinder =
      MapBinder.newMapBinder(binder(), String.class,
        Method.class, new RegistrarImpl(RegistrarType.EVIDENCE));

    evidenceInterfaceMapBinder
      .addBinding(CASEEVIDENCE.ASSET)
        .toInstance(FactoryMethodHelper.getNewInstanceMethod(
          curam.seg.evidence.entity.fact.AssetFactory.class));
  }
}
```

3.8.2 Legacy Evidence Registrar

The legacy mechanism for registration of evidence entities is still supported. i.e. using the Application Properties to specify the factories to populate a hashmap of the hook classes. The factory code will not change in order to maintain backward compatibility but all out of the box, legacy implementations have been deprecated.

3.9 Custom Hooks

As the Evidence Controller functionality is generic to all evidence solutions, the only way to facilitate an organization's unique requirements is by the provision of hooks where custom logic can be located in order to extend the core solution. Call outs to these hooks, or extension points, are made within the Evidence Controller maintenance functions.

3.9.1 Evidence Controller Hook

Evidence Controller Hook is the evidence infrastructure class which contains the extension points for the evidence maintenance pattern. An example of a hook in this class is postRemoveEvidence. A call is made to this function inside the Evidence Controller removeEvidence operation. Customers must override the hook with their custom version if they want to perform post remove evidence processing.

3.9.2 Evidence Controller Hook Registrar & Manager

Following on from the Evidence Registrar and the underlying Dependency Injection pattern, a similar approach has been taken for the registration of the Evidence Controller Hook class. An Evidence Controller Hook Registrar interface is shipped as part of the evidence infrastructure. As before, at runtime, the Evidence Controller invokes the Registrar's register method which performs the dependency injection of the associated custom Evidence Controller Hook. This is the class which will have extended the out-of-the-box Evidence Controller Hook and overridden the methods being customized. This "injector" class is located through runtime configuration where the injector class itself is referred to as the "Evidence Controller Hook Registrar".

The dependency injection involves two steps. First, a custom Evidence Controller Hook Registrar, which implements the Evidence Controller Hook Registrar interface, must be located and the Registrar then invoked to register the customized hook class. For example, the product type and custom Evidence Controller Hook class pairing will be entered into a hashmap and then the class looked up via the product type when it's required. In order to locate the Evidence Controller Hook Registrar, its class name must be configured using the environment variable "curam.case.evidencecontrollerhook.registrars". Note: additional entries need to be added to this environment variable in a comma delimited format.

The implementation of the Registrar's register method must reference the customized Evidence Controller Hook class. Doing this in code, rather than as configuration, provides a compile time check that the referenced class exists. The existence of the Registrar, though, is only ascertained from the provided configuration, and may result in a runtime failure if the application is mis-configured.

The Evidence Controller Hook Manager class manages the static initialization of the Evidence Controller Hook mapping as well as the retrieval of the subclass of the Evidence Controller Hook. If no subclass is found, the out-of-the-box version of the Evidence Controller Hook class is returned.

Chapter 4. Participant Evidence Integration

4.1 Overview

Evidence is the term used for data in the calculation of eligibility and entitlement. Participant data is also regarded as evidence, a concern's date of birth for example, but in the past it wasn't always treated as classic evidence. It is obviously correct for a concern's date of birth to be maintained within the Participant Manager rather than being stored on a separate evidence entity, i.e. one that is interfaced to the Evidence API, but it must also be propagated across all cases belonging to the concern and any changes in such evidence must trigger reassessment.

- A modification applied to Participant data will automatically apply to all cases using this data
- Modifying such data will trigger reassessment of all cases using this data

The following Core Participant entities have been integrated with Evidence:

- Address
- AlternateID
- AlternateName
- BankAccount
- Citizenship
- ConcernRole
- ConcernRoleRelationship
- Education
- Employer
- Employment
- EmploymentWorkHour
- Foreign Residency
- Person
- ProspectEmployer
- ProspectPerson

4.2 Integration of Participant Data as Evidence

Participant Evidence Integration is available out of the box but, like evidence, it requires a certain amount of configuration. If the configuration is not carried out, then all newly integrated Participant evidence will not integrate with the Evidence API. It will, however, continue to function as it always has. Once configured, the Participant evidence will be linked to one or more cases via an Evidence Descriptor. As in the case of classic evidence, the Evidence Descriptor can be associated with either an Integrated Case or a Product Delivery.

The required configuration links the Participant evidence types to the Integrated Case(s) or Product(s) that will use them. Such data is stored on the AdminICEvidenceLink and ProductEvidenceLink respectively. Participant data that will be stored at the Integrated Case level needs to be configured on the AdminICEvidenceLink entity whereas Participant evidence that will be used by a Product needs to be configured on the ProductEvidenceLink entity.

4.3 Administration

4.3.1 AdminICEvidenceLink

Every integrated case type that wants to integrate the available 15 entities as evidence will need to insert an entry into the AdminICEvidenceLink table. This table must link evidenceMetadataID (from EvidenceMetadata table) and adminIntegratedCaseID (from AdminIntegratedCase table) for each participant entity required as evidence and for each integrated case type.

4.3.2 ProductEvidenceLink

Every product delivery case type that wants to integrate the available 15 entities as evidence will need to insert an entry into the ProductEvidenceLink table. This table must link evidenceMetadataID (from EvidenceMetadata table) and productID (from Product table) for each participant entity required as evidence and for each product type.

4.4 Integrating new Participant entities as Evidence

Integrating new, or existing, Participant entities with Evidence requires a number of steps. As mentioned above, meta-data needs to be configured for Integrated Case types and Product types. As well as this, other infrastructural support needs to be implemented by a developer in order for the integration to work.

4.4.1 Implementing the ParticipantEvidenceInterface

A Participant entity being integrated into the Evidence solution must implement the ParticipantEvidenceInterface. This means that the entity will need to implement the following functions:

- calcAttributionDatesForCase
- getDetailsForListDisplay
- getEndDate
- getStartDate
- insertEvidence
- insertEvidenceOnModify
- modifyEvidence
- readAllByParentID
- readEvidence
- selectForValidation
- validate
- checkForReassessment
- createSnapshot
- getChangedAttributeList
- readAllByConcernRoleID
- removeEvidence

4.4.2 Register entity in a Registrar Module

Participant entities being integrated to Evidence need to be registered via a Registrar Module as outlined in 3.8.1, “Evidence Registrar Module,” on page 23. The out of the box participant evidence types has been configured in CoreRegistrarModule. This binds the evidence type to it's entity. These map bindings are loaded at runtime and are used by the Evidence Controller when looking up the appropriate evidence entity for a given type, i.e. the entity that has implemented the ParticipantEvidenceInterface.

4.4.3 Applying Participant Evidence to all Cases

A new hook class ApplyChangesForEvidence has been added.

The new ApplyChangesForEvidence class represents a hook which can be overridden by custom code. The ApplyChangesForEvidence.isApplyChangesAutomatedForEvidence method is called from Evidence Controller to decide whether reassessment needs to be triggered when evidence is applied. The default implementation defaults to false and therefore the user will have to manually apply the changes on the associated cases. If the solutions wish to customize, the implementers should use ProductHookRegistrar.registerApplyChangesHooks method to add details of the hooks to use for applying changes. The static map attribute, applyChangesHookMap present in ProductHookManager class is used to store pairs of product type and the name of the class that implements the hook for that product type. The method ProductHookManager.getApplyChangesHook gets the implementation subclass of the ApplyChangesForEvidence class for the specified product type. The method EvidenceController.applyParticipantEvodence has been updated to obtain product delivery and product details for the case and then call ProducHookManager.getApplyChangesHook to obtain correct instance of the ApplyChangesForEvidence class for the given product.

4.4.4 Modifications required to existing business processes

In all places where there are existing calls to insert, modify, and less frequently, remove methods, the code needs to be updated to invoke the EvidenceController as well as the insert, modify and remove methods as appropriate. An example of how an insert works with Evidence is shown below:

```
// insert new citizenship entry
citizenshipObj.insert(citizenshipDtls);
```

Figure 5. Before

```
//
// Call the EvidenceController object and insert evidence
// Evidence descriptor details
EvidenceDescriptorInsertDtls evidenceDescriptorInsertDtls =
  new EvidenceDescriptorInsertDtls();
evidenceDescriptorInsertDtls.participantID =
  details.concernRoleID;
evidenceDescriptorInsertDtls.evidenceType =
  CASEEVIDENCE.CITIZENSHIP;
evidenceDescriptorInsertDtls.receivedDate =
Date.getCurrentDate();

// Evidence Interface details
EIEvidenceInsertDtls eiEvidenceInsertDtls =
  new EIEvidenceInsertDtls();
eiEvidenceInsertDtls.descriptor.assign(
  evidenceDescriptorInsertDtls);
eiEvidenceInsertDtls.descriptor.participantID =
  citizenshipDtls.concernRoleID;
eiEvidenceInsertDtls.evidenceObject =
  citizenshipDtls;

// EvidenceController business object
curam.core.sl.infrastructure.impl.EvidenceControllerInterface
  evidenceControllerObj =
  (curam.core.sl.infrastructure.impl.EvidenceControllerInterface)
  curam.core.sl.infrastructure.fact.EvidenceControllerFactory
  .newInstance();

// Insert the evidence
EIEvidenceKey eiEvidenceKey =
  evidenceControllerObj.insertEvidence(eiEvidenceInsertDtls);
```

Figure 6. After

4.5 Sequence Diagrams for Participant evidence

The development, both client and server, of creating and modifying evidence operations are outlined here:

4.5.1 Create Participant Evidence Sequence Diagram

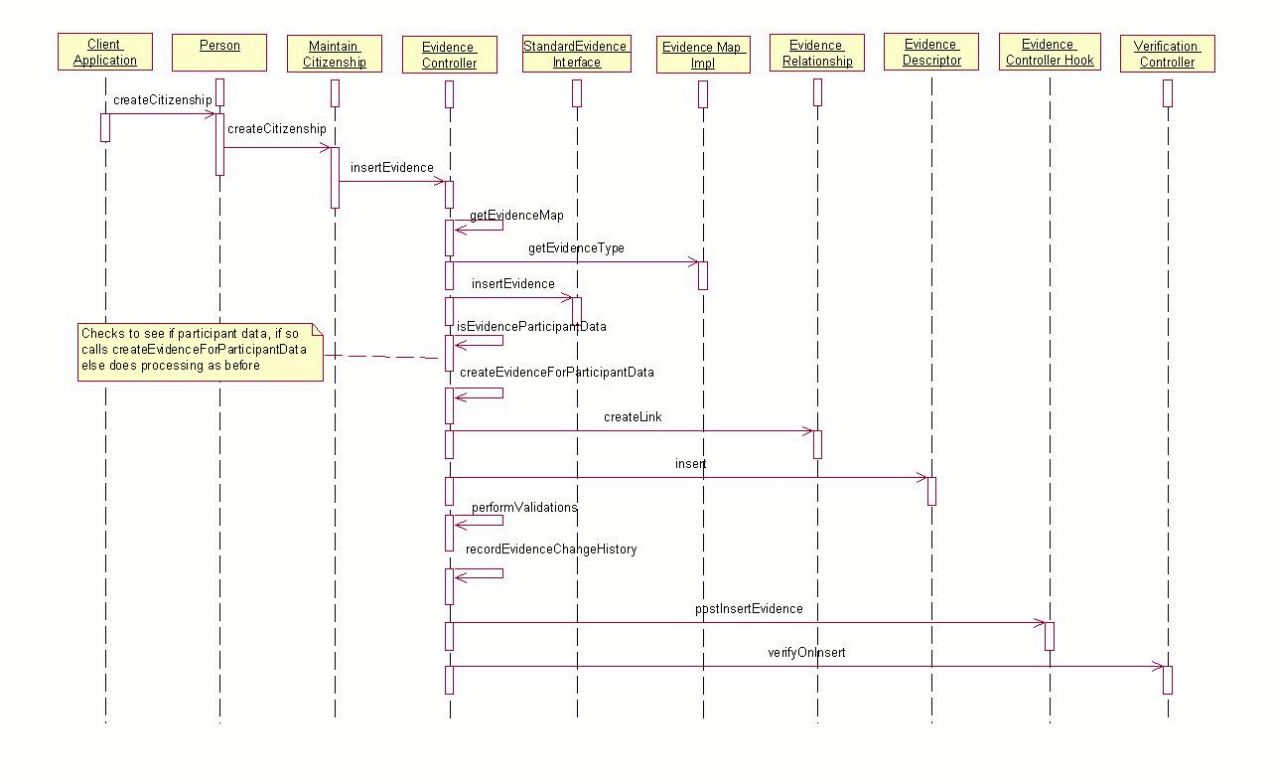

Figure 7. Participant Evidence Sequence

4.5.2 Specific Processing For Participant Data when Creating Evidence

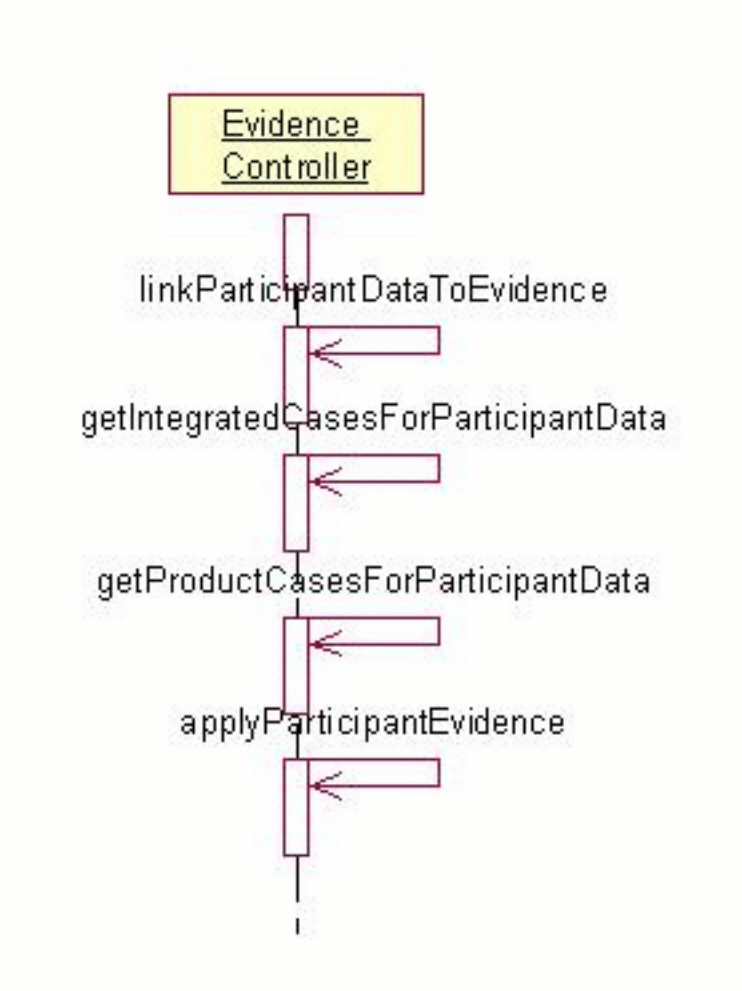

Figure 8. Evidence Sequence Diagram

4.5.3 Modify Participant Evidence Sequence Diagram

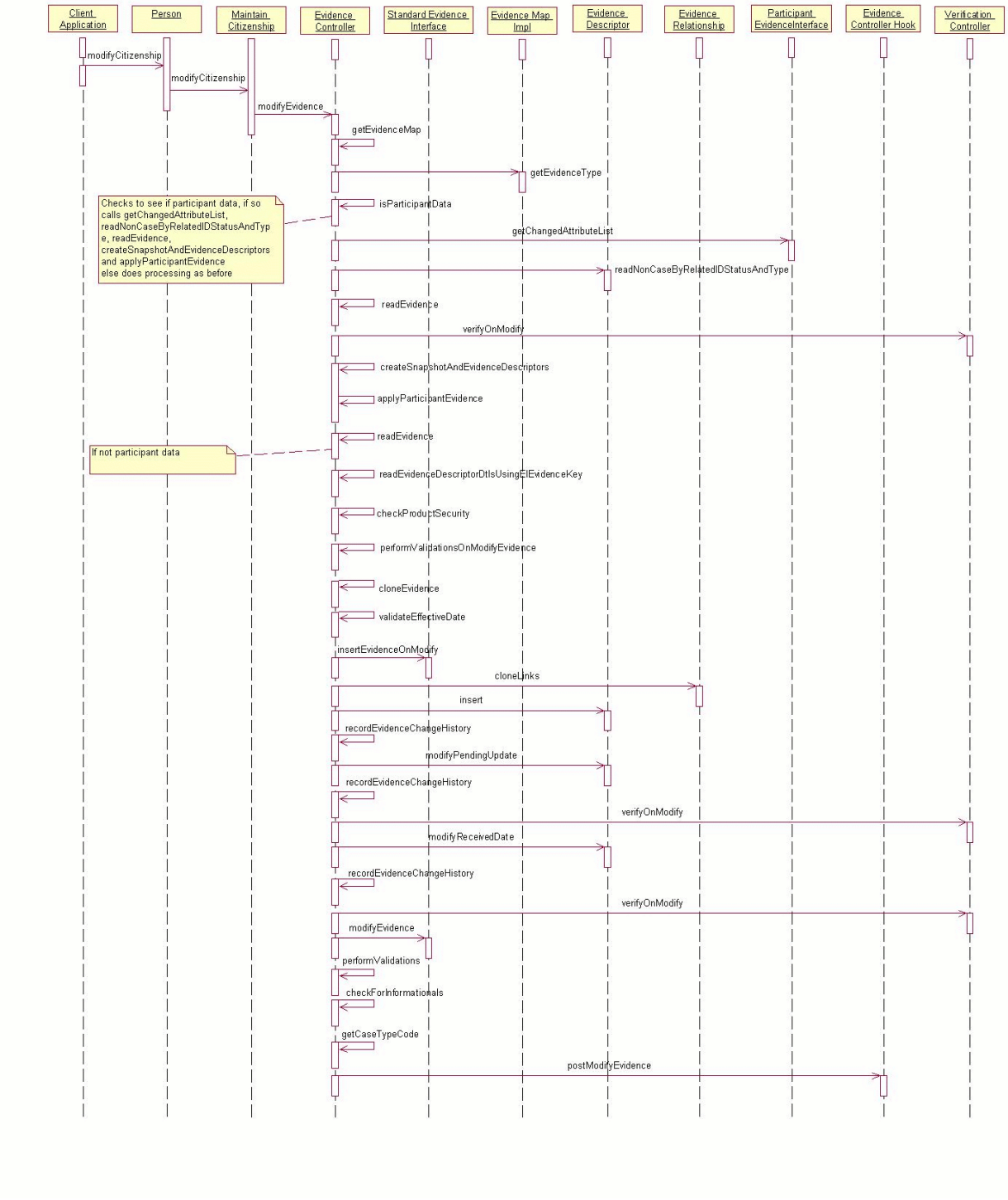

Figure 9. Modify participant

Appendix. Appendix

A.1 Appendix A

A.1.1 Conditional Verification

Conditional Verifications is a feature, wherein, the flexibility is provided to determine if verification is applicable for evidence through programmatic support as opposed to manually means. The programmatic support is encompassed through rule-class implementations and verification for a piece of evidence is determined based on a set of conditions. The Verification Engine will check the conditions specified, at the time of adding or modifying evidence but will create an outstanding verification only when a condition that has been defined is met and not every time a verifiable data item is added or modified. The conditions can range from conditions against the value of the verifiable data item to more complex conditions where the values of a set of dependent evidences determine whether or not verification is required.

A.1.2 Rule Artifacts supplied by Verification framework

To facilitate integration between Verification framework and the rule implementations supplied by other components, the framework supplies core Rule Artifacts. These artifacts contain abstract rule classes that other components rule implementations must adhere to. This section identifies and details such low level Rule Artifacts which will be supplied as part of Verification framework.

A.1.3 Rule Sets

The rule set *'VerificationRuleSet'* is available as part of Verification framework. This rule set holds all the framework's artifacts such as the rule classes and the data container classes.

A.1.4 Rule Classes

The following rule classes are available as part of *'VerificationRuleSet'*. The purpose of these rule classes are explained in the corresponding sections.

VerificationDeterminator

VerificationDeterminatorResult

VerificationDeterminatorParams

A.1.5 Verification Determinator

The business logic that determines whether conditional verification is required for particular evidence type goes in this rule class. Components creating rule implementations must adhere to the specification by directly/indirectly extending this class. The following attributes are available in this rule class.

S.No	Rule attribute name	Type	Purpose
1	*determine*	*A.1.6, "Verification Determinator Result"*	The implementation will contain the business logic that determines the output of conditional verification. A value of 'TRUE' indicates to the evidence framework that verifications are not applicable for the evidence, whereas 'FALSE' denotes that verifications need to be explicitly added.
2	*verificationDeterminatorParams*	*A.1.7, "Verification Determinator Params"*	This attribute is populated by the Conditional verifications framework and contains the values for all the input parameters for a particular instance.

A.1.6 Verification Determinator Result

This rule class is a data container whose purpose is to store the results of business logic in the A.1.5, "Verification Determinator," on page 33. Currently this class has two attributes,

result - a boolean that states whether verification is required or not for a given evidence

reason - a codetable value from VerificationSkippedReason, which contains the values of reason for which the conditional verification is not applicable

It is the responsibility of the rule implementations to create/populate these attribute so that the verification framework, after examining the state of the attribute, can take appropriate business decisions.

A.1.7 Verification Determinator Params

While determining whether conditional verification is required or not, the framework will supply various input parameters to the rule implementation classes for various calculation purposes such as the evidence that is getting currently edited, the associated case identifier for the evidence etc. Please refer the following table for complete details of the input parameters.

S.No	Property Name	Data Type	Description
1	*verifiableDataItemName*	String	Represents the name of the *'Verifiable Data Item'* such as *'Person Income'*, *'Date Of Birth'* etc. The value comes from the code table 'VerifiableItemName'
2	*evidenceDescriptorID*	Number	The unique identifier of the evidence record in question
3	*caseID*	Number	The unique identifier of the case with which the evidence is associated

A.1.8 New Propagator

Verifications are applicable to active evidences as well as to evidences which are in 'in-edit' state. A new propagator – **ActiveInEditEvidenceRowRuleObjectPropagator** is provided for this very purpose, which will propagate both these evidence type. It is recommended to use this new propagator to propagate the evidences to the rule data objects that are used in the conditional verification implementation classes.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources.

IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing

application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/us/en/copytrade.shtml.

Java and all Java-based trademarks and logos are registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.

Printed in USA