IBM Cúram Social Program Management

Guía del desarrollador de pruebas de persona y candidato de Cúram

Versión 6.0.5

IBM Cúram Social Program Management

Guía del desarrollador de pruebas de persona y candidato de Cúram

Versión 6.0.5

Nota

Antes de utilizar esta información y el producto al que hace referencia, lea la información que figura en el apartado "Avisos" en la página 39

Revisado: mayo de 2013

Esta edición se aplica a IBM Cúram Social Program Management v6.0 5 y a todos los releases subsiguientes hasta que se indique lo contrario en nuevas ediciones.

Contenido

Figuras

Tablas

Capítulo 1. Introducción

1.1 Finalidad

La finalidad de esta guía es proporcionar una descripción técnica de alto nivel de las pruebas de persona o candidato y de sus componentes. Esta guía también describe las opciones de personalización disponibles y los puntos de ampliación y proporciona instrucciones sobre cómo implementar estas personalizaciones.

1.2 Público al que va dirigida

Esta guía está destinada a los desarrolladores y arquitectos que tienen la intención de implementar una solución de pruebas de persona o candidato.

Importante: Esta guía sólo es aplicable para los lectores que utilizan la aplicación de participante con pruebas dinámicas de persona y candidato.

1.3 Requisitos previos

En el documento se supone que el lector está familiarizado con lo siguiente:

- *Guía sobre pruebas de Cúram*
- *Guía para participantes de Cúram*
- *Guía de configuración de pruebas dinámicas de Curam*
- *Google Guice*

1.4 Capítulos de esta guía

La siguiente lista describe los capítulos de esta guía:

Visión general de pruebas de persona o candidato
Este capítulo proporciona una visión general de alto nivel de los aspectos técnicos clave de las pruebas de la persona o del candidato.

Diseño de soluciones de pruebas de persona o candidato
En este capítulo se proporcionan consideraciones de diseño que deben tenerse en cuenta al diseñar una solución de pruebas de persona/candidato.

Correlaciones de datos
Este capítulo describe la correlación de datos de los tipos de pruebas dinámicas proporcionados con la aplicación con las tablas de base de datos heredadas.

Personalización de pruebas de persona/candidato
En este capítulo se describen las opciones de personalización y los puntos de ampliación disponibles para las pruebas de persona/candidato.

Capítulo 2. Visión general de pruebas de persona o candidato

2.1 Datos de persona/candidato como pruebas

Parte de la información almacenada sobre personas y candidatos se retiene como prueba, que puede compartirse entre casos en el sistema. Diversos componentes y tecnologías de Cúram vienen juntos para permitir el almacenamiento de pruebas de persona y candidato y el flujo de estas pruebas a través del sistema:

- Las pruebas dinámicas de Cúram se utilizan para almacenar los datos de persona/candidato capturados y realizar validaciones básicas.
- Las Reglas de Cúram Express se utilizan para ejecutar validaciones complejas en los datos capturados.
- El Agente de pruebas de Cúram se puede utilizar opcionalmente para traspasar los datos entre casos.
- Las Verificaciones de Cúram se pueden utilizar opcionalmente para aplicar verificaciones a los datos capturados cuando se traspasan entre casos.

En función de los requisitos de negocio, es posible que sea necesario cierto nivel de configuración y/o personalización. En este capítulo se describe a un nivel elevado la forma en que el sistema gestiona los datos de persona/candidato e identifica los puntos en los que pueden ser necesarias la configuración y/o personalización.

Nota: Se deberá tener en cuenta de manera cuidadosa el comportamiento de negocio necesario del sistema durante la fase de diseño de la aplicación de Cúram. El punto de partida para desarrollar conocimientos sobre cómo se debe configurar un sistema para soportar los requisitos empresariales deben ser la Guía para participantes de Cúram y la Guía sobre pruebas de Cúram.

2.2 ¿Cómo se gestionan las pruebas de persona/candidato?

La gestión de los datos de persona y candidato como pruebas se basa en los puntos clave siguientes:

- Por consiguiente, cada persona y candidato tiene un caso de persona o candidato asociado (Caso de datos de participante) creado 'de forma encubierta' al registrarse.
- Los datos de persona y candidato se almacenan como pruebas en las tablas de pruebas dinámicas y se describen mediante los tipos de pruebas dinámicas que están asociados con la persona y/o candidato.
- Los datos registrados como pruebas se replican de nuevo en las tablas de base de datos heredadas; por consiguiente, las tablas de base de datos heredadas deben estar en sincronización con las pruebas dinámicas.
- Al editar un registro de persona o candidato, los datos se recuperan de las tablas de pruebas dinámicas y se graban en ellas (y de nuevo, se vuelven a replicar en las tablas de base de datos heredadas).
- En algunos casos, las pantallas de aplicación y el proceso seguirán leyendo en las tablas de base de datos heredadas.
- Los tipos de caso de persona y candidato se pueden configurar para que los datos de participante se traspasen entre casos, utilizando el agente de pruebas de Cúram.

2.2.1 Tipos de pruebas de persona o candidato

Se proporcionan diversos tipos de pruebas dinámicas que se asocian con las personas y los candidatos participantes. Estos tipos de pruebas y sus atributos no se deben eliminar o desasociarse de la persona y el candidato.

Cuando sea necesario gestionar datos adicionales como pruebas de persona y candidato, se pueden crear tipos de pruebas nuevos, tal como se describe en la publicación Guía de configuración de pruebas dinámicas de Cúram, y asociar con la persona y el candidato en el componente de administración.

Asimismo, se pueden añadir atributos nuevos pueden a los tipos de pruebas dinámicas existentes. Cuando los datos que se están añadiendo a los tipos de pruebas nuevos o existentes ya están presentes en una tabla de base de datos heredada existente, se debe realizar un trabajo de personalización adicional:

- El código que replica los datos de las tablas de pruebas dinámicas en las tablas de base de datos heredados (el 'replicador') debe ampliarse para replicar los datos adicionales que se están almacenando como pruebas.
- Cuando ya existen datos en las tablas de base de datos heredadas, estos datos se deben copiar en la(s) tabla(s) equivalente(s) de pruebas dinámicas. El código que realiza dicha operación (el 'conversor') se debe ampliar para convertir los datos adicionales en pruebas.

Se proporcionan más detalles e implementaciones de ejemplo de ambos en el capítulo 'Personalización de pruebas de persona/candidato'.

2.2.2 Validaciones de pruebas

Las pruebas de la persona y del candidato se validan cuando se crean y se editan. Estas validaciones se implementan de una de dos maneras:

- Utilizando la funcionalidad de validaciones del Editor de pruebas dinámicas. Por ejemplo, validaciones de campo obligatorio.
- Utilizando conjuntos de reglas de Cúram Express. Por ejemplo, validaciones de pruebas cruzadas.

Cuando se añaden atributos o tipos de pruebas dinámicas nuevos, los clientes deben utilizar uno de estos mecanismos para añadir las validaciones necesarias. Esto se describe más detalladamente en la publicación Guía de configuración de pruebas dinámicas de Cúram.

Cuando las pruebas se traspasan a la persona y al candidato, estas validaciones no se comprueban. Las pruebas traspasadas se deben aceptar siempre para evitar que se pierdan, porque no existe ningún concepto de 'pruebas entrantes' para una persona y un candidato. Cuando se especifican las pruebas de persona/candidato, éstas se validan inmediatamente. No obstante, las pruebas traspasadas de otro caso se aceptan y activan automáticamente, incluso si las comprobaciones de validación fallan. Para otros tipos de caso, cuando las pruebas de persona/candidato se traspasan al caso, una anomalía de validación impedirá que se activen las pruebas.

2.2.3 Uso compartido de pruebas

La infraestructura de pruebas ofrece la posibilidad de compartir pruebas entre una persona/candidato, casos de solicitud y casos en curso. El Agente de pruebas habilita y arbitra este uso compartido de pruebas. El uso compartido de pruebas es unidireccional y se realiza por tipo de pruebas. Esto significa que destinos diferentes pueden recibir y compartir un tipo de pruebas de distintas maneras. Si es necesario, es posible que un tipo de caso reciba pruebas compartidas, pero no pueda compartir sus propias pruebas. Existen tres funciones principales que desencadenarán que el agente de pruebas difunda las pruebas:

- Cuando se añade una persona nueva a un caso de destino. Por ejemplo, cuando se han configurado pruebas de persona/candidato, por ejemplo pruebas de identificación, para el uso compartido en un caso integrado y se añade una persona a un caso integrado, el agente de pruebas comprueba primero si esa persona tiene pruebas de persona/candidato. Si se encuentran pruebas, el agente de pruebas comprueba a continuación las pruebas de identificación activas y las comparte en el caso integrado.
- Cuando se realizan cambios de pruebas en una caso de origen. Por ejemplo, cuando se realizan cambios en las pruebas de identificación de una persona, el agente de pruebas comparte esos cambios en el caso integrado.
- Cuando se crea un nuevo caso de destino. Por ejemplo, siempre que se cree un caso integrado nuevo, el agente de pruebas buscará pruebas de identificación de la persona/candidato para compartirlas. Si se encuentran estas pruebas, el agente de pruebas comparte las pruebas de identificación en el caso integrado.

Para obtener información más detallada sobre el agente de pruebas, consulte la Guía del agente de pruebas de Cúram.

Capítulo 3. Diseño de soluciones de pruebas de persona o candidato

Al diseñar una solución de pruebas de persona o candidato, el diseñador debe tener en cuenta los datos, la estructura, las restricciones y el flujo de dichos datos en el sistema.

3.1 Datos: Tipos de pruebas dinámicas

3.1.1 Estructura

Las pruebas de persona/candidato se almacenan principalmente como pruebas dinámicas y las estructuras de datos que las representan son tipos de pruebas dinámicas. Los tipos de pruebas dinámicas definen los datos, el tipo y comportamiento, como por ejemplo volatilidad, atributos calculados, etc. Una vez que se ha definido un nuevo tipo de pruebas dinámicas, éste se debe activar y asociar con los tipos de casos relevantes, las personas y los candidatos. Encontrará información adicional sobre cómo definir tipos de pruebas dinámicas se pueden encontrar en la publicación Guía de configuración de pruebas dinámicas de Cúram.

Aspectos a tener en cuenta

- ¿Varían las pruebas con el tiempo?
- ¿Es recíproco el tipo de pruebas? Si es así, el tipo de pruebas debe tener atributos de participante y participante relacionado.
- ¿En qué tipos de caso deben estar disponibles las pruebas?
- Considere la posibilidad de convertir el tipo de pruebas en 'Preferido', si se va a utilizar de manera habitual. Permitirá a los asistentes sociales crear pruebas rápidamente para tipos de pruebas registradas con frecuencia.

3.1.2 Restricciones

3.1.2.1 Validaciones

En el Editor de pruebas dinámicas se proporcionan diversas validaciones estándares que se usan con frecuencia en el proceso de pruebas. Se pueden incluir validaciones más complejas, por ejemplo validaciones de pruebas cruzadas, utilizando Reglas de Cúram Express. Se puede encontrar más información sobre las validaciones en la publicación Guía de configuración de pruebas dinámicas de Cúram.

Aspectos a tener en cuenta

- ¿Qué validaciones son necesarias para garantizar la integridad de los datos?
- Cuando se entran pruebas de persona/candidato, éstas se validan inmediatamente; no obstante, las pruebas traspasadas de otro caso se aceptan y activan automáticamente, incluso si las comprobaciones de validación fallan. Para otros tipos de caso, cuando las pruebas de persona/candidato se traspasan al caso, una anomalía de validación impedirá que se activen las pruebas.
- Incluya las validaciones necesarias para imponer restricciones de sucesión.
- Intente utilizar patrones de validación estándares siempre que sea posible. Sólo se deberán desarrollar conjuntos de reglas de validación si no se pueden implementar utilizando validaciones estándares.
- Si se desarrollan conjuntos de reglas de validación para validaciones más complejas, sea consciente de cómo se realiza la recuperación de datos. Si realiza incorrectamente, esto puede tener un impacto significativo en el rendimiento.

Importante: Los procesos del sistema se basan en las validaciones que se suministran con la aplicación y está permitido eliminar o alterar estas validaciones.

3.1.2.2 Verificaciones

Este apartado es sólo aplicable a los lectores que tienen licencia para utilizar el componente de verificaciones.

La verificación es el proceso de comprobación de la precisión de las pruebas. La verificación de pruebas puede adoptar diversas formas; se pueden proporcionar mediante documentos, por ejemplo certificados de nacimiento o extractos bancarios, o medios verbales, por ejemplo llamadas telefónicas. Cuando se capturan pruebas, se invoca el motor de verificación para determinar si las pruebas requieren verificación.

Nota: Con la excepción de las pruebas traspasadas a un registro de persona o candidato, las pruebas no se pueden activar hasta que se han satisfecho todos los requisitos de verificación obligatorios.

Para obtener más información sobre las verificaciones y su configuración, consulte la publicación Cúram Verification Guide.

Aspectos a tener en cuenta

- ¿Necesitan verificación estas pruebas?
- ¿Cuáles son las reglas respecto a la verificación?
- ¿Qué información necesita proporcionar el cliente?

3.2 Flujo: Agente de pruebas

El Agente de pruebas es un mecanismo que permite compartir pruebas en todo el sistema. Cuando el agente de pruebas difunde pruebas en un registro de persona/candidato, las pruebas se aceptan y se activan automáticamente en el registro de persona/candidato, de modo que el usuario no tiene que aceptar y activar manualmente las pruebas. Para obtener más información sobre el agente de pruebas y las opciones de configuración, consulte la publicación Guía del Agente de pruebas de Cúram. Para conocer la propuesta recomendada para el traspaso, consulte la Guía sobre pruebas de Cúram.

Aspectos a tener en cuenta

- ¿Se utiliza el mismo tipo de pruebas en más de un tipo de caso? Si es así, ¿se deberán comunicar a otros casos los cambios realizados en estas pruebas?
- ¿Se deberá configurar el caso de destino para que acepte automáticamente los cambios o se deberá forzar la intervención del asistente social para que éste decida si se deben aceptar estas pruebas entrantes?
- Para que el proceso del sistema funcione correctamente, es esencial que los datos de persona/candidato registrados fuera del gestor de participantes se compartan con el gestor de participantes.

3.3 Reglas de Cúram Express: Cálculos de elegibilidad/derecho de un caso

Las áreas donde se utilizan Reglas de Cúram Express para leer datos de participante de tablas de base de datos heredadas para realizar cálculos de elegibilidad/derecho de un caso se deben analizar y decidir qué origen de datos es el más adecuado. Existen tres opciones, cada una de las cuales tiene sus propias ventajas y limitaciones:

- Leer los datos de participante de las pruebas dinámicas almacenadas por el gestor de participantes.
- Leer los datos de participante que se han traspasado a casos
- Continuar leyendo en las tablas heredadas

3.3.1 Leer los datos de participante de las pruebas dinámicas almacenadas por el gestor de participantes

Aspectos a tener en cuenta

- Utilización desde el origen de datos principal
- Los cambios en las pruebas producen nuevos cálculos inmediatos
- No hay ninguna oportunidad para que el asistente social realice una revisión

3.3.2 Leer los datos de participante que se han traspasado a casos

Aspectos a tener en cuenta

- Ésta es la opción recomendada para un desarrollo nuevo
- Los cambios sólo tendrán efecto cuando se activen las pruebas
- El tipo de pruebas debe estar configurado para traspasarse al caso

3.3.3 Continuar leyendo en las tablas heredadas

Aspectos a tener en cuenta

- Esta opción debe elegirse con cuidado y sólo se recomienda para actualizar clientes

Capítulo 4. Correlaciones de datos de tipos de pruebas dinámicas

Las tablas siguientes muestran las correlaciones de datos de los tipos de pruebas dinámicas con las tablas de base de datos heredadas.

Nota: Los replicadores realizan esta correlación y los conversores realizan la correlación inversa.

4.1 Dirección

Tabla 1. Correlación de dirección

Atributo de pruebas dinámicas	Columna de la base de datos
participant	ConcernRoleAddress.concernRoleID (se calcula utilizando caseParticipantRoleID)
address	Address.addressData
fromDate	ConcernRoleAddress.startDate
toDate	ConcernRoleAddress.endDate
addressType	ConcernRoleAddress.typeCode
comments	ConcernRoleAddress.comments

4.2 Cuenta bancaria

Tabla 2. Correlación de cuenta bancaria

Atributo de pruebas dinámicas	Columna de la base de datos
participant	ConcernRoleBankAccount.concernRoleID (se calcula utilizando caseParticipantRoleID)
accountName	BankAccount.name
accountNumber	BankAccount.accountNumber
accountType	BankAccount.typeCode
sortCode	BankAccount.bankSortCode
fromDate	BankAccount.startDate
toDate	BankAccount.endDate
accountStatus	BankAccount.bankAccountStatus
jointAccountInd	BankAccount.jointAccountInd
comments	BankAccount.comments

4.3 Nacimiento y defunción

Tabla 3. Correlación de nacimiento y defunción

Atributo de pruebas dinámicas	Columna de la base de datos
person	Person/ProspectPerson.concernRoleID (se calcula utilizando caseParticipantRoleID)
birthLastName	Person/ProspectPerson.personBirthName

Tabla 3. Correlación de nacimiento y defunción (continuación)

Atributo de pruebas dinámicas	Columna de la base de datos
mothersBirthLastName	Person/ProspectPerson.motherBirthSurname
dateOfBirth	Person/ProspectPerson.dateOfBirth
dateOfDeath	Person/ProspectPerson.dateOfDeath
comments	N/D

4.4 Preferencias de contacto

Tabla 4. Correlación de preferencias de contacto

Atributo de pruebas dinámicas	Columna de la base de datos
participant	ConcernRole.concernRoleID (se calcula utilizando caseParticipantRoleID)
preferredLanguage	ConcernRole.preferredLanguage
preferredCommunication	ConcernRole.prefCommMethod
comments	N/D

4.5 Dirección de correo electrónico

Tabla 5. Correlación de dirección de correo electrónico

Atributo de pruebas dinámicas	Columna de la base de datos
participant	ConcernRoleEmailAddress.concernRoleID (se calcula utilizando caseParticipantRoleID)
emailAddress	EmailAddress.emailAddress
fromDate	ConcernRoleEmailAddress.startDate
toDate	ConcernRoleEmailAddress.endDate
emailAddressType	ConcernRoleEmailAddress.typeCode
comments	EmailAddress.comments

4.6 Género

Tabla 6. Correlación de género

Atributo de pruebas dinámicas	Columna de la base de datos
person	Person/ProspectPerson.concernRoleID (se calcula utilizando caseParticipantRoleID)
gender	Person/ProspectPerson.gender
comments	N/D

4.7 Identificación

Tabla 7. Correlación de identificación

Atributo de pruebas dinámicas	Columna de la base de datos
participant	ConcernRoleAlternateID.concernRoleID (se calcula utilizando caseParticipantRoleID)

Tabla 7. Correlación de identificación (continuación)

Atributo de pruebas dinámicas	Columna de la base de datos
alternateID	ConcernRoleAlternateID.alternateID
altIDType	ConcernRoleAlternateID.typeCode
fromDate	ConcernRoleAlternateID.startDate
toDate	ConcernRoleAlternateID.endDate
comments	ConcernRoleAlternateID.comments

4.8 Nombre

Tabla 8. Correlación de nombre

Atributo de pruebas dinámicas	Columna de la base de datos
participant	AlternateName.concernRoleID (se calcula utilizando caseParticipantRoleID)
title	AlternateName.title
firstName	AlternateName.firstForename
middleName	AlternateName.otherForename
lastName	AlternateName.surname
suffix	AlternateName.nameSuffix
initials	AlternateName.initials
nameType	AlternateName.nameType
comments	AlternateName.comments

4.9 Número de teléfono

Tabla 9. Correlación de número de teléfono

Atributo de pruebas dinámicas	Columna de la base de datos
participant	ConcernRolePhoneNumber.concernRoleID (se calcula utilizando caseParticipantRoleID)
phoneCountryCode	PhoneNumber.phoneCountryCode
phoneAreaCode	PhoneNumber.phoneAreaCode
phoneNumber	PhoneNumber.phoneNumber
phoneExtension	PhoneNumber.phoneExtension
fromDate	ConcernRolePhoneNumber.startDate
toDate	ConcernRolePhoneNumber.endDate
phoneType	ConcernRolePhoneNumber.typeCode
comments	PhoneNumber.comments

4.10 Relación

Tabla 10. Correlación de relación

Atributo de pruebas dinámicas	Columna de la base de datos
participant	ConcernRoleRelationship.concernRoleID (se calcula utilizando caseParticipantRoleID)
relatedParticipant	ConcernRoleRelationship.relConcernRoleID
fromDate	ConcernRoleRelationship.startDate
toDate	ConcernRoleRelationship.endDate
relationshipType	ConcernRoleRelationship.relationshipType
endReason	ConcernRoleRelationship.relEndReasonCode
comments	ConcernRoleRelationship.comments

Capítulo 5. Personalización de pruebas de persona/candidato

5.1 Introducción

En este capítulo se describen las opciones de personalización y los puntos de ampliación disponibles para las pruebas de persona/candidato. Algunos o todos estos puntos pueden ser aplicables en función de las personalizaciones y configuraciones existentes. Hay cinco áreas principales que deben tenerse en cuenta; se indican a continuación:

- Replicadores
- Conversores
- Selección de información principal
- Pruebas recíprocas
- Propietario de caso de datos de participante

Cada una de estas áreas se describen en detalle y también se proporcionan ejemplos. **Tenga en cuenta que estos son sólo ejemplos.**

5.2 Replicadores

5.2.1 ¿Qué es un replicador?

Un replicador refleja los cambios en las pruebas en las tablas heredadas pertinentes, a efectos de compatibilidad con versiones anteriores. El replicador toma los detalles de pruebas dinámicas y los convierte en una estructura que contiene los detalles que se deben almacenar en las tablas heredadas. A continuación, estos detalles se graban en las tablas de base de datos pertinentes, asegurando de este modo que la información de las tablas heredadas está en sincronización con el origen de datos primario, las pruebas dinámicas. Se proporcionan implementaciones de replicador predeterminadas para cada uno de los tipos de pruebas de persona/candidato. Estas implementaciones predeterminadas contienen puntos de ampliación para permitir la réplica en los campos personalizados, lo cual se describe en el siguiente apartado.

Nota: Sólo se utiliza la última versión en un conjunto de sucesión para replicar datos en las tablas heredadas.

5.2.2 ¿Por qué ampliar un replicador?

En los casos en los que se han ampliado las tablas de base de datos heredadas puede que sea necesario ampliar un replicador.

5.2.3 Ampliación de replicador

Es posible ampliar los replicadores suministrados con la aplicación para permitir la replicación de pruebas de persona o candidato en las columnas de base de datos personalizadas. Hay interfaces disponibles para cada tipo de pruebas proporcionado y se pueden encontrar en el paquete curam.pdc.impl, que se muestra a continuación. Se pueden grabar implementaciones personalizadas que utilicen estas interfaces, en función del tipo de pruebas.

Interfaces de ampliador de replicador

- PDCAddressReplicatorExtender
- PDCAlternateIDReplicatorExtender
- PDCAlternateNameReplicatorExtender
- PDCBankAccountReplicatorExtender

- PDCBirthAndDeathReplicatorExtender
- PDCContactPreferencesReplicatorExtender
- PDCEmailAddressReplicatorExtender
- PDCGenderReplicatorExtender
- PDCPhoneNumberReplicatorExtender
- PDCRelationshipsReplicatorExtender

La mayoría de las interfaces tienen un método `assignDynamicEvidenceToExtendedDetails`. Éste acepta dos parámetros:

- dynamicEvidenceDataDetails: detalles de pruebas dinámicas
- details: estructura que contiene los detalles ampliados para la tabla heredada

`PDCBirthAndDeathReplicatorExtender` y `PDCGenderReplicatorExtender` tienen dos métodos, `assignDynamicEvidenceToExtendedPersonDetails` y `assignDynamicEvidenceToExtendedProspectPersonDetails`. `assignDynamicEvidenceToExtendedPersonDetails` acepta dos parámetros:

- dynamicEvidenceDataDetails: detalles de pruebas dinámicas
- details: estructura que contiene los detalles de la persona ampliados para la tabla heredada

`assignDynamicEvidenceToExtendedProspectPersonDetails` también acepta dos parámetros:

- dynamicEvidenceDataDetails: detalles de pruebas dinámicas
- details: estructura que contiene los detalles del candidato ampliados para la tabla heredada

5.2.4 Ejemplo: Implementación de un ampliador de replicador de pruebas de persona/candidato

El ejemplo siguiente describe cómo ampliar un replicador para correlacionar pruebas de persona/candidato con campos personalizados. Este ejemplo proporciona una implementación muy básico de una ampliación de `PDCPhoneNumberReplicatorExtender`. En este escenario, la tabla PhoneNumber se ha ampliado y contiene un campo personalizado 'phoneProvider'. La configuración de pruebas dinámicas para el número de teléfono también contiene un atributo 'phoneProvider'. En este ejemplo se supone que la estructura `ParticipantPhoneDetails` ya se ha ampliado para incluir el campo personalizado. Para obtener más información sobre la configuración de pruebas dinámicas, consulte la publicación Guía de configuración de pruebas dinámicas de Cúram. La responsabilidad de la implementación de ampliación de replicador personalizada es correlacionar los datos de pruebas dinámicas con el atributo de estructura que representa el atributo 'phoneProvider' en la tabla PhoneNumber ampliada.

Nota: Una correlación de datos es lo único que es necesario; la implementación predeterminada realizar la réplica real de los datos.

Los pasos implicados en la ampliación de un replicador son:

- Proporcionar una implementación de ampliación de replicador que correlacionará los datos personalizados con la tabla heredada.
- Añadir un enlace a la nueva implementación de ampliación de replicador.

5.2.4.1 Paso 1: Proporcionar una implementación de ampliación de replicador

El primer paso es proporcionar una implementación nueva que implemente la interfaz de ampliación de réplica pertinente del tipo de pruebas y correlacione los datos personalizados con la tabla heredada. El fragmento de código siguiente muestra una implementación personalizada de `PDCPhoneNumberReplicatorExtender`. Simplemente asigna el valor del atributo de pruebas dinámicas al

atributo de estructura phoneProvider. Entonces esta información se insertará junto con los demás atributos de pruebas dinámicas a través de la implementación predeterminada de PDCPhoneNumberReplicatorExtender.

```
public class SampleReplicatorExtenderImpl
  implements PDCPhoneNumberReplicatorExtender {

  public void assignDynamicEvidenceToExtendedDetails(
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails,
    ParticipantPhoneDetails details)
    throws AppException, InformationalException {

    details.phoneProvider =
    dynamicEvidenceDataDetails.getAttribute("phoneProvider").getValue();
  }
}
```

5.2.4.2 Paso 2: Añadir un enlace a la nueva implementación de ampliación de replicador

Se utilizan enlaces Guice para registrar la implementación.

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Registrar la implementación de ampliación de replicador
  Multibinder<PDCPhoneNumberReplicatorExtender> sampleReplicatorExtender =
  Multibinder.newSetBinder(binder(), PDCPhoneNumberReplicatorExtender.class);

  sampleReplicatorExtender.addBinding().to(SampleReplicatorExtenderImpl.class);
 }
}
```

Nota: Los módulos Guice nuevos deben registrarse añadiendo una fila en la tabla de base de datos ModuleClassName. Consulte la publicación Persistence Cookbook para obtener más información.

5.2.5 ¿Por qué implementar un replicador?

En los casos en los que se han presentado nuevos tipos de pruebas dinámicas, puede que sea necesario implementar un replicador nuevo.

5.2.6 Implementación de un replicador

Los replicadores se pueden desarrollar fácilmente para proveer escenarios como nuevos tipos de pruebas dinámicas. En el apartado siguiente se proporciona un ejemplo detallado que describe los pasos y artefactos necesarios para lograr que un replicador nuevo esté activo y en ejecución. Las implementaciones de replicador se invocan a través de un mecanismo basado en sucesos. Cuando se activan las pruebas dinámicas después de una inserción, modificación o eliminación, se genera un suceso. Para los tipos de pruebas nuevas, es necesario desarrollar un escucha de sucesos para escuchar este suceso e invocar el proceso de réplica; esto se describirá más detalladamente más adelante en este capítulo. El apartado siguiente muestra cómo implementar un replicador.

5.2.7 Ejemplo: Implementación de un replicador de pruebas de persona/candidato

El ejemplo siguiente describe cómo implementar un replicador. En este escenario, el ejemplo de residencia en el extranjero se ha configurado como un nuevo tipo de pruebas dinámicas. Para obtener más información sobre cómo configurar un nuevo tipo de pruebas, consulte la Guía de configuración de pruebas dinámicas de Cúram. El nuevo tipo de pruebas del ejemplo de residencia en el extranjero tiene los atributos siguientes:

- participant: ID de rol de participante en el caso de la persona/candidato para la que se están entrando las pruebas

- country: el país de residencia
- fromDate: la fecha en la que se ha iniciado la residencia
- toDate: la fecha en la que ha finalizado la residencia
- reason: la razón de la residencia en este país

Se ha supuesto que este tipo de pruebas dinámicas se ha activado y está configurado para utilizarse con personas o candidatos. Hasta ahora, la información del ejemplo de residencia en el extranjero se almacenaba como pruebas estáticas en la tabla de base de datos SampleForeignResidency. Ahora es necesario almacenar esta información como pruebas dinámicas. Es posible que se necesite un replicador nuevo para replicar los cambios de pruebas en la tabla de base de datos heredada a fin de que esta tabla esté sincronizada con las pruebas dinámicas.

Los pasos implicados en la implementación de un replicador son:
- Proporcionar una interfaz de replicador para el tipo de pruebas dinámicas.
- Proporcionar una implementación de replicador que replicará las pruebas dinámicas en la tabla de base de datos heredada.
- Implementar un escucha de sucesos que desencadenará la réplica.
- Añadir un enlace a la nueva implementación de escucha de sucesos.

5.2.7.1 Paso 1: Proporcionar una interfaz de replicador

La nueva interfaz de replicador debe contener tres métodos:

`replicateInsertEvidence` que replica las pruebas del Ejemplo de residencia en el extranjero insertadas activadas en la tabla de base de datos heredada del ejemplo de residencia en el extranjero. Acepta un parámetro:
- evidenceDescriptorDtls: detalles de descriptor de pruebas activadas

`replicateModifyEvidence`: que replica las pruebas del Ejemplo de residencia en el extranjero activadas modificadas en la tabla de base de datos heredada del ejemplo de residencia en el extranjero. Acepta dos parámetros:
- evidenceDescriptorDtls: detalles de descriptor de pruebas activadas
- previousActiveEvidDescriptorDtls: detalles de descriptor de las pruebas que estaban activas antes de la modificación

`replicateRemoveEvidence`: que replica las pruebas del Ejemplo de residencia en el extranjero activadas eliminadas en la tabla de base de datos heredada de ejemplo de residencia en el extranjero. Acepta un parámetro:
- evidenceDescriptorDtls: detalles de descriptor de pruebas activadas

```
@ImplementedBy(SampleForeignResidencyReplicatorImpl.class)
public interface SampleForeignResidencyReplicator {

  public void replicateInsertEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException;

  public void replicateModifyEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls,
    final EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException;

  public void replicateRemoveEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException;
}
```

5.2.7.2 Paso 2: Proporcionar una implementación de replicador

La implementación de replicador debe proporcionar implementaciones para los tres métodos descritos en la sección anterior. Estos métodos deben convertir los datos de pruebas dinámicos en datos adecuados para grabarse en tablas de base de datos heredadas y actualizar las tablas heredadas para este tipo de pruebas.

```
public class SampleForeignResidencyReplicatorImpl
  implements SampleForeignResidencyReplicator {

  protected SampleForeignResidencyReplicatorImpl() {
  }

  public void replicateInsertEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    SampleForeignResidency sampleForeignResidencyObj =
   SampleForeignResidencyFactory.newInstance();
    SampleForeignResidencyDtls sampleForeignResidencyDtls =
   new SampleForeignResidencyDtls();
    UniqueID uniqueIDObj = UniqueIDFactory.newInstance();

    EvidenceControllerInterface evidenceControllerObj =
      (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
    eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
    eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

    EIEvidenceReadDtls eiEvidenceReadDtls =
      evidenceControllerObj.readEvidence(eiEvidenceKey);

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
      (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();

    sampleForeignResidencyDtls.startDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("fromDate"));

    sampleForeignResidencyDtls.endDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("toDate"));

    sampleForeignResidencyDtls.reasonCode =
      dynamicEvidenceDataDetails.getAttribute("reason").getValue();

    sampleForeignResidencyDtls.concernRoleID = evidenceDescriptorDtls.participantID;
    sampleForeignResidencyDtls.foreignResidencyID = uniqueIDObj.getNextID();
    sampleForeignResidencyDtls.statusCode = RECORDSTATUS.NORMAL;

    sampleForeignResidencyObj.insert(sampleForeignResidencyDtls);
  }

  public void replicateModifyEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls,
    final EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException {

    List<SampleForeignResidencyKey> sampleForeignResidencyKeyList =
      new ArrayList<SampleForeignResidencyKey>();

    SampleForeignResidencyDtls sampleForeignResidencyDtls =
   new SampleForeignResidencyDtls();
```

```
 EvidenceControllerInterface evidenceControllerObj =
   (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

 EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
 eiEvidenceKey.evidenceID = previousActiveEvidDescriptorDtls.relatedID;
 eiEvidenceKey.evidenceType = previousActiveEvidDescriptorDtls.evidenceType;

 EIEvidenceReadDtls eiEvidenceReadDtls =
   evidenceControllerObj.readEvidence(eiEvidenceKey);

 DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
   (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

 sampleForeignResidencyDtls.countryCode =
   dynamicEvidenceDataDetails.getAttribute("country").getValue();

 sampleForeignResidencyDtls.startDate =
(Date) DynamicEvidenceTypeConverter.convert(
   dynamicEvidenceDataDetails.getAttribute("fromDate"));

 sampleForeignResidencyDtls.endDate =
(Date) DynamicEvidenceTypeConverter.convert(
   dynamicEvidenceDataDetails.getAttribute("toDate"));

 sampleForeignResidencyDtls.reasonCode =
dynamicEvidenceDataDetails.getAttribute("reason").getValue();

 SampleForeignResidency sampleForeignResidencyObj =
SampleForeignResidencyFactory.newInstance();

 SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
   new SampleForeignResidencyReadMultiKey();
 sampleForeignResidencyReadMultiKey.concernRoleID =
previousActiveEvidDescriptorDtls.participantID;

 SampleForeignResidencyReadMultiDtlsList sampleForeignResidencyReadMultiDtlsList =
   sampleForeignResidencyObj.searchByConcernRole(sampleForeignResidencyReadMultiKey);

 for (SampleForeignResidencyReadMultiDtls sampleForeignResidencyReadMultiDtls :
sampleForeignResidencyReadMultiDtlsList.dtls) {

   if ((sampleForeignResidencyReadMultiDtls.countryCode.equals(
  sampleForeignResidencyDtls.countryCode))
     && (sampleForeignResidencyReadMultiDtls.reasonCode.equals(
sampleForeignResidencyDtls.reasonCode))) {

     SampleForeignResidencyKey sampleForeignResidencyKey = new SampleForeignResidencyKey();
     sampleForeignResidencyKey.sampleForeignResidencyID =
 sampleForeignResidencyReadMultiDtls.sampleForeignResidencyID;

     sampleForeignResidencyKeyList.add(sampleForeignResidencyKey);
   }
 }

 for (SampleForeignResidencyKey sampleForeignResidencyKey : sampleForeignResidencyKeyList) {

   sampleForeignResidencyDtls = new SampleForeignResidencyDtls();

   eiEvidenceKey = new EIEvidenceKey();
   eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
   eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

   eiEvidenceReadDtls = evidenceControllerObj.readEvidence(eiEvidenceKey);

   dynamicEvidenceDataDetails =
     (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;
```

```
    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();

    sampleForeignResidencyDtls.startDate = (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("fromDate"));

    sampleForeignResidencyDtls.endDate = (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("toDate"));

    sampleForeignResidencyDtls.reasonCode =
      dynamicEvidenceDataDetails.getAttribute("reason").getValue();

    sampleForeignResidencyDtls.concernRoleID = evidenceDescriptorDtls.participantID;

    SampleForeignResidencyDtls sampleForeignResidencyReadDtls =
   sampleForeignResidencyObj.read(sampleForeignResidencyKey);

    sampleForeignResidencyReadDtls.assign(sampleForeignResidencyDtls);

    sampleForeignResidencyObj.modify(sampleForeignResidencyKey, sampleForeignResidencyReadDtls);
  }
}

public void replicateRemoveEvidence(
  final EvidenceDescriptorDtls evidenceDescriptorDtls)
  throws AppException, InformationalException {

  List<SampleForeignResidencyKey> sampleForeignResidencyKeyList =
 new ArrayList<SampleForeignResidencyKey>();

  SampleForeignResidencyDtls sampleForeignResidencyDtls =
 new SampleForeignResidencyDtls();

  EvidenceControllerInterface evidenceControllerObj =
    (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

  EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
  eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
  eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

  EIEvidenceReadDtls eiEvidenceReadDtls =
    evidenceControllerObj.readEvidence(eiEvidenceKey);

  DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
    (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

  sampleForeignResidencyDtls.countryCode =
    dynamicEvidenceDataDetails.getAttribute("country").getValue();

  sampleForeignResidencyDtls.startDate =
 (Date) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails.getAttribute("fromDate"));

  sampleForeignResidencyDtls.endDate =
 (Date) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails.getAttribute("toDate"));

  sampleForeignResidencyDtls.reasonCode =
 dynamicEvidenceDataDetails.getAttribute("reason").getValue();

  SampleForeignResidency sampleForeignResidencyObj =
 SampleForeignResidencyFactory.newInstance();

  SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
    new SampleForeignResidencyReadMultiKey();
  sampleForeignResidencyReadMultiKey.concernRoleID =
 evidenceDescriptorDtls.participantID;
```

```
    SampleForeignResidencyReadMultiDtlsList sampleForeignResidencyReadMultiDtlsList =
      sampleForeignResidencyObj.searchByConcernRole(sampleForeignResidencyReadMultiKey);

    for (SampleForeignResidencyReadMultiDtls sampleForeignResidencyReadMultiDtls :
   sampleForeignResidencyReadMultiDtlsList.dtls) {

      if ((sampleForeignResidencyReadMultiDtls.countryCode.equals(
     sampleForeignResidencyDtls.countryCode))
        && (sampleForeignResidencyReadMultiDtls.reasonCode.equals(
  sampleForeignResidencyDtls.reasonCode))) {

        SampleForeignResidencyKey sampleForeignResidencyKey = new SampleForeignResidencyKey();
        sampleForeignResidencyKey.sampleForeignResidencyID =
    sampleForeignResidencyReadMultiDtls.sampleForeignResidencyID;

        sampleForeignResidencyKeyList.add(sampleForeignResidencyKey);
      }
    }

    for (SampleForeignResidencyKey sampleForeignResidencyKey : sampleForeignResidencyKeyList) {

      sampleForeignResidencyDtls = sampleForeignResidencyObj.read(sampleForeignResidencyKey);
      sampleForeignResidencyDtls.statusCode = RECORDSTATUS.CANCELLED;
      sampleForeignResidencyObj.modify(sampleForeignResidencyKey, sampleForeignResidencyDtls);
    }
  }
}
```

5.2.7.3 Paso 3: Implementar un escucha de sucesos

Es necesario implementar un nuevo escucha de sucesos para escuchar los sucesos generados del tipo del ejemplo de residencia en el extranjero que se producen como resultado de la activación de pruebas. Este escucha debe implementar la interfaz curam.pdc.impl.PDCEvents y proporcionar implementaciones para los tres métodos. Aquí es donde se puede iniciar el proceso de réplica así como cualquier otro proceso personalizado que pueda tener que producirse.

```
public class SampleForeignResidencyEventsListener
  implements PDCEvents {

  @Inject
  private SampleForeignResidencyReplicator sampleForeignResidencyReplicator;

  public void insertedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    if (evidenceDescriptorDtls.evidenceType.equals("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateInsertEvidence(evidenceDescriptorDtls);
    }
  }

  public void modifiedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls,
    EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException {

    if (evidenceDescriptorDtls.evidenceType.equals("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateModifyEvidence(evidenceDescriptorDtls,
      previousActiveEvidDescriptorDtls);
    }
  }

  public void removedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {
```

```
 if (evidenceDescriptorDtls.evidenceType.equals("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateRemoveEvidence(evidenceDescriptorDtls);
    }
  }
}
```

5.2.7.4 Paso 4: Añadir un enlace a la nueva implementación de escucha de sucesos

Se utilizan enlaces Guice para registrar la implementación.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Registrar el escucha de sucesos
    Multibinder<PDCEvents> sampleEventListeners =
      Multibinder.newSetBinder(binder(), PDCEvents.class);

    sampleEventListeners.addBinding().to(
      SampleForeignResidencyEventsListener.class);
  }
}
```

Nota: Los módulos Guice nuevos deben registrarse añadiendo una fila en la tabla de base de datos ModuleClassName. Consulte la publicación Persistence Cookbook para obtener más información.

5.3 Conversores

5.3.1 ¿Qué es un conversor?

Un conversor es un mecanismo para convertir datos de persona/candidato heredados en pruebas dinámicas. Cuando las tablas heredadas se han llenado de datos externos a la aplicación, mediante el uso de herramientas como Cúram Data Manager (archivos DMX), se pueden utilizar conversores para convertir estos datos en pruebas dinámicas. Se proporcionan implementaciones de conversor predeterminadas para cada uno de los tipos de pruebas de persona/candidato. Estas implementaciones predeterminadas contienen puntos de ampliación para permitir la conversión de los campos personalizados, lo cual se describe en el siguiente apartado.

5.3.2 ¿Por qué ampliar un conversor?

En los casos en los que se han ampliado las tablas de base de datos heredadas puede que sea necesario ampliar un conversor. En general, los conversores sólo se utilizan en un entorno de desarrollo o para la herramienta de actualización y no se deben utilizar como parte del proceso diario.

5.3.3 Ampliación de conversor

Los conversores proporcionados con la aplicación se pueden ampliar para permitir convertir las columnas de base de datos personalizadas en pruebas dinámicas de persona/candidato. Hay interfaces disponibles para cada tipo de pruebas original, que se pueden encontrar en el paquete curam.pdc.impl, y se listan a continuación. Se pueden grabar implementaciones personalizadas que utilicen estas interfaces, en función del tipo de pruebas.

Interfaces de ampliación de conversor (llenado)

- PDCAddressEvidencePopulator
- PDCAlternateIDEvidencePopulator
- PDCAlternateNameEvidencePopulator
- PDCBankAccountEvidencePopulator
- PDCBirthAndDeathEvidencePopulator

- PDCContactPreferencesEvidencePopulator
- PDCEmailAddressEvidencePopulator
- PDCGenderEvidencePopulator
- PDCPhoneNumberEvidencePopulator
- PDCRelationshipsEvidencePopulator

La mayoría de las interfaces tienen un método `populate`. Éste acepta diferentes parámetros en función de los tipos de pruebas.

`PDCBirthAndDeathEvidencePopulator` y `PDCGenderEvidencePopulator`, las interfaces tienen dos métodos, `populatePerson` y `populateProspectPerson`.

`populatePerson` acepta cuatro parámetros:
- concernRoleKey: identificador exclusivo para el rol de asunto con el que se están relacionando estas pruebas
- caseIDKey: identificador exclusivo del caso de datos de participante
- personDtls: estructura que contiene los detalles de la persona ampliados de la tabla heredada
- dynamicEvidenceDataDetails: detalles de pruebas dinámicas

`populateProspectPerson` también acepta cuatro parámetros:
- concernRoleKey: identificador exclusivo para el rol de asunto con el que se están relacionando estas pruebas
- caseIDKey: identificador exclusivo del caso de datos de participante
- prospectPersonDtls: estructura que contiene los detalles de candidato ampliados de la tabla heredada
- dynamicEvidenceDataDetails: detalles de pruebas dinámicas

5.3.4 Ejemplo: Implementación de un poblador de pruebas de persona persona/candidato

El ejemplo siguiente describe cómo ampliar un conversor para correlacionar columnas de base de datos personalizadas con pruebas de persona/candidato. Este ejemplo proporciona una implementación muy básica de una ampliación en `PDCPhoneNumberEvidencePopulator`. En este escenario, la tabla PhoneNumber se ha ampliado y contiene una columna personalizada 'phoneProvider'. La configuración de pruebas dinámicas para el número de teléfono también contiene un atributo 'phoneProvider'. La responsabilidad de la implementación de poblador personalizada es convertir el atributo de estructura que representa el atributo 'phoneProvider' en la tabla PhoneNumber ampliada en datos de pruebas dinámicas. Para obtener más información sobre la configuración de pruebas dinámicas, consulte la publicación Guía de configuración de pruebas dinámicas de Cúram.

Nota: La conversión de datos es lo único que es necesario; los conversores predeterminados se encargarán del almacenamiento real de las pruebas dinámicas.

Los pasos implicados en la ampliación de un conversor son:
- Proporcionar una implementación de poblador que convertirá el campo personalizado de la tabla heredada en datos de pruebas dinámicas.
- Añadir un enlace a la nueva implementación de poblador.

5.3.4.1 Paso 1: Proporcionar una implementación de poblador

El primer paso es proporcionar una implementación nueva que implemente la interfaz de poblador pertinente para el tipo de pruebas y convierta el campo personalizado de la tabla heredada en pruebas dinámicas. El fragmento de código siguiente muestra la implementación personalizada para `PDCPhoneNumberEvidencePopulator`: simplemente convierte el atributo de estructura phoneProvider en el

atributo equivalente de pruebas dinámicas. Estas pruebas dinámicas se almacenan entonces junto con los demás atributos de pruebas dinámicas a través de la implementación de conversor predeterminado.

```
public class SamplePopulatorImpl
  implements PDCPhoneNumberEvidencePopulator {

  public void populate(
    ConcernRoleKey concernRoleKey, CaseIDKey caseIDKey,
    ConcernRolePhoneNumberDtls concernRolePhoneNumberDtls,
    PhoneNumberDtls phoneNumberDtls,
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails)
    throws AppException, InformationalException {

    DynamicEvidenceDataAttributeDetails phoneProvider =
      dynamicEvidenceDataDetails.getAttribute("phoneProvider");

    DynamicEvidenceTypeConverter.setAttribute(phoneProvider,
      phoneNumberDtls.phoneProvider);
  }
}
```

5.3.4.2 Añadir un enlace a la nueva implementación de poblador

Se utilizan enlaces Guice para registrar la implementación.

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Registrar la implementación de poblador
  Multibinder<PDCPhoneNumberEvidencePopulator> samplePopulator =
  Multibinder.newSetBinder(binder(), PDCPhoneNumberEvidencePopulator.class);

  samplePopulator.addBinding().to(SamplePopulatorImpl.class);
  }
}
```

Nota: Los módulos Guice nuevos deben registrarse añadiendo una fila en la tabla de base de datos ModuleClassName. Consulte la publicación Persistence Cookbook para obtener más información.

5.3.5 ¿Por qué implementar un conversor?

En los casos en los que se han presentado nuevos tipos de pruebas dinámicas, puede que sea necesario implementar un conversor nuevo. En general, los conversores sólo se utilizan en un entorno de desarrollo o para la herramienta de actualización y no se deben utilizar como parte del proceso diario.

5.3.6 Implementación de un conversor

Las implementaciones de conversor se pueden desarrollar utilizando la interfaz `PDCConverter`. La interfaz `PDCConverter` se puede encontrar en el paquete curam.pdc.impl. Esta interfaz tiene un método `storeEvidence`. Acepta dos parámetros:

- concernRoleKey: identificador exclusivo del rol de asunto
- caseIDKey: identificador exclusivo del Caso de datos de participante.

El apartado siguiente muestra cómo implementar un conversor.

5.3.7 Ejemplo: Implementación de un conversor de pruebas de persona/candidato

El ejemplo siguiente describe cómo implementar un conversor. En este escenario, el ejemplo de residencia en el extranjero se ha configurado como un nuevo tipo de pruebas dinámicas. Para obtener más información sobre cómo configurar un nuevo tipo de pruebas, consulte la Guía de configuración de pruebas dinámicas de Cúram. El nuevo tipo de pruebas del ejemplo de residencia en el extranjero tiene los atributos siguientes:

- participant: ID de rol de participante en el caso de la persona/candidato para la que se están entrando las pruebas
- country: el país de residencia
- fromDate: la fecha en la que se ha iniciado la residencia
- toDate: la fecha en la que ha finalizado la residencia
- reason: la razón de la residencia en este país

Se ha supuesto que este tipo de pruebas dinámicas se ha activado y está configurado para utilizarse con personas o candidatos. La información del ejemplo de residencia en el extranjero se almacenaba anteriormente como pruebas estáticas en la tabla de base de datos SampleForeignResidency. Ahora es necesario almacenar esta información como pruebas dinámicas. Se necesita un nuevo conversor que tomará esta información de la tabla heredada y la convertirá y almacenará como pruebas dinámicas.

Los pasos implicados en la implementación de un conversor son:
- Proporcionar una implementación de conversor que convertirá los datos heredados en pruebas dinámicas.
- Añadir un enlace a la nueva implementación de conversor.

5.3.7.1 Paso 1: Proporcionar una implementación de conversor

El fragmento de código siguiente muestra la implementación de `PDCConverter`. Recupera toda la información del ejemplo de residencia en el extranjero de una persona/un candidato de la tabla SampleForeignResidency heredada, convierte esta información en una estructura de datos de pruebas dinámica y almacena la información resultante.

```
public class SampleForeignResidencyConverterImpl
  implements PDCConverter {

  @Inject
  private EvidenceTypeDefDAO etDefDAO;

  @Inject
  private EvidenceTypeVersionDefDAO etVerDefDAO;

  public void storeEvidence(ConcernRoleKey concernRoleKey, CaseIDKey caseIDKey)
    throws AppException, InformationalException {

    PDCCaseIDCaseParticipantRoleID pdcCaseIDCaseParticipantRoleID =
      new PDCCaseIDCaseParticipantRoleID();

    ParticipantDataCase participantDataCaseObj = ParticipantDataCaseFactory.newInstance();
    pdcCaseIDCaseParticipantRoleID.caseID =
      participantDataCaseObj.getParticipantDataCase(concernRoleKey).caseID;

    CaseIDTypeCodeKey caseIDTypeCodeKey = new CaseIDTypeCodeKey();
    caseIDTypeCodeKey.caseID = pdcCaseIDCaseParticipantRoleID.caseID;
    caseIDTypeCodeKey.typeCode = CASEPARTICIPANTROLETYPE.PRIMARY;

    pdcCaseIDCaseParticipantRoleID.caseParticipantRoleID =
      CaseParticipantRoleFactory.newInstance().readByCaseIDAndTypeCode(caseIDTypeCodeKey).dtls.caseParticipantRoleID;

    SampleForeignResidency sampleForeignResidencyObj = SampleForeignResidencyFactory.newInstance();

    SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
      new SampleForeignResidencyReadMultiKey();
    sampleForeignResidencyReadMultiKey.concernRoleID = concernRoleKey.concernRoleID;

    SampleForeignResidencyReadMultiDtlsList sampleForeignResidencyList =
      sampleForeignResidencyObj.searchByConcernRole(sampleForeignResidencyReadMultiKey);

   for (SampleForeignResidencyReadMultiDtls sampleForeignResidencyReadMultiDtls : sampleForeignResidencyList.dtls) {

      final EvidenceTypeKey eType = new EvidenceTypeKey();
```

```
      eType.evidenceType = "SampleForeignResidency";

      EvidenceTypeDef evidenceType =
        etDefDAO.readActiveEvidenceTypeDefByTypeCode(eType.evidenceType);

      EvidenceTypeVersionDef evTypeVersion =
        etVerDefDAO.getActiveEvidenceTypeVersionAtDate(evidenceType, Date.getCurrentDate());

      DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
        DynamicEvidenceDataDetailsFactory.newInstance(evTypeVersion);

      DynamicEvidenceDataAttributeDetails participant =
        dynamicEvidenceDataDetails.getAttribute("participant");

      DynamicEvidenceTypeConverter.setAttribute(participant,
        pdcCaseIDCaseParticipantRoleID.caseParticipantRoleID);

      DynamicEvidenceDataAttributeDetails country =
        dynamicEvidenceDataDetails.getAttribute("country");

      DynamicEvidenceTypeConverter.setAttribute(country,
        sampleForeignResidencyReadMultiDtls.countryCode);

      DynamicEvidenceDataAttributeDetails fromDate =
        dynamicEvidenceDataDetails.getAttribute("fromDate");

      DynamicEvidenceTypeConverter.setAttribute(fromDate, sampleForeignResidencyReadMultiDtls.startDate);

      DynamicEvidenceDataAttributeDetails endDate =
        dynamicEvidenceDataDetails.getAttribute("toDate");

      DynamicEvidenceTypeConverter.setAttribute(endDate, sampleForeignResidencyReadMultiDtls.endDate);

      DynamicEvidenceDataAttributeDetails reasonCode =
        dynamicEvidenceDataDetails.getAttribute("reason");

      DynamicEvidenceTypeConverter.setAttribute(reasonCode, sampleForeignResidencyReadMultiDtls.reasonCode);

      EvidenceControllerInterface evidenceControllerObj =
        (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

      EvidenceDescriptorInsertDtls evidenceDescriptorInsertDtls = new EvidenceDescriptorInsertDtls();
      evidenceDescriptorInsertDtls.participantID = concernRoleKey.concernRoleID;
      evidenceDescriptorInsertDtls.evidenceType = eType.evidenceType;
      evidenceDescriptorInsertDtls.receivedDate = curam.util.type.Date.getCurrentDate();
      evidenceDescriptorInsertDtls.caseID = pdcCaseIDCaseParticipantRoleID.caseID;

      EIEvidenceInsertDtls eiEvidenceInsertDtls = new EIEvidenceInsertDtls();

      eiEvidenceInsertDtls.descriptor.assign(evidenceDescriptorInsertDtls);
      eiEvidenceInsertDtls.descriptor.participantID = concernRoleKey.concernRoleID;
      eiEvidenceInsertDtls.descriptor.changeReason = EVIDENCECHANGEREASON.REPORTEDBYCLIENT;
      eiEvidenceInsertDtls.evidenceObject = dynamicEvidenceDataDetails;

      evidenceControllerObj.insertEvidence(eiEvidenceInsertDtls);
    }
  }
}
```

5.3.7.2 Paso 2: Añadir un enlace a la nueva implementación de conversor

Se utilizan enlaces Guice para registrar la implementación.

```
public class SampleModule extends AbstractModule {

  public void configure() {

  // Registrar la implementación de conversor
```

```
  Multibinder<PDCConverter> sampleForeignResidencyConverter =
    Multibinder.newSetBinder(binder(), PDCConverter.class);

  sampleForeignResidencyConverter.addBinding().to(SampleForeignResidencyConverterImpl.class);
  }
}
```

Nota: Los módulos Guice nuevos deben registrarse añadiendo una fila en la tabla de base de datos ModuleClassName. Consulte la publicación Persistence Cookbook para obtener más información.

5.4 Selección de información principal

Las entidades del gestor de participantes heredadas tienen la noción de indicadores principales, donde los usuarios pueden especificar qué cuenta bancaria, número de teléfono, etc. representa los datos principales donde se crean las pruebas. Esto no sucede con los tipos de pruebas dinámicas. El usuario no especifica el registro principal sino que, en su lugar, existe un algoritmo en segundo plano que calcula cuál debe ser el registro principal. Estos algoritmos, que se basan en una estrategia de negocio definida y se pueden modificar, se describen detalladamente en el apartado siguiente.

5.4.1 ¿Por qué cambiar la selección de información principal?

Dado que la identificación del registro principal ya está controlada por el usuario, puede que sea necesario modificar este proceso de selección, si no se prefiere la estrategia de negocio predeterminada.

5.4.2 Cambio de la selección de información principal

Las estrategias que determinan qué datos deben seleccionarse como información principal pueden modificarse mediante el uso de las interfaces de manejador principal predeterminado. Se define una interfaz para cada tipo de pruebas dinámicas proporcionado que tiene un identificador principal en su tabla heredada, que se pueden encontrar en el paquete curam.pdc.impl y se listan a continuación.

Las implementaciones de manejador principal se invocan a través de un mecanismo basado en sucesos. Cuando se activan las pruebas dinámicas después de una inserción, modificación o eliminación, se genera un suceso. Para los tipos de pruebas nuevas, es necesario desarrollar un escucha de sucesos para escuchar este suceso e invocar el algoritmo apropiado que determinará los datos principales. Esto se describirá más detalladamente posteriormente en este capítulo. El apartado siguiente muestra cómo implementar un manejador principal.

Interfaces de manejador principal

- PDCPrimaryAddressHandler
- PDCPrimaryAlternateIDHandler
- PDCPrimaryAlternateNameHandler
- PDCPrimaryBankAccountHandler
- PDCPrimaryEmailAddressHandler
- PDCPrimaryPhoneNumberHandler

5.4.3 Cambio de la selección de ejemplo de información principal

El ejemplo siguiente describe cómo implementar un manejador principal. En este escenario, la estrategia de negocio definida para seleccionar un número de teléfono principal consiste en seleccionar el número de teléfono con la última fecha de inicio.

Los pasos implicados en la implementación de un manejador principal son:

- Proporcionar una implementación de manejador principal que identifique el registro principal.
- Añadir un enlace a la nueva implementación de manejador principal.

5.4.3.1 Paso 1: Proporcionar una implementación de manejador principal

El primer paso es proporcionar una nueva implementación que implemente la interfaz de manejador primario pertinente para el tipo de pruebas e identifique el registro principal. El fragmento de código siguiente muestra una implementación para PDCPrimaryPhoneNumberHandler. Simplemente toma el número de teléfono con la última fecha de inicio y lo establece como registro principal.

```
public class SamplePrimaryPhoneNumberHandlerImpl
  implements PDCPrimaryPhoneNumberHandler {

  protected SamplePrimaryPhoneNumberHandlerImpl() {
  }

  public void setPrimaryPhoneNumber(EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    ConcernRoleKey concernRoleKey = new ConcernRoleKey();
    concernRoleKey.concernRoleID = evidenceDescriptorDtls.participantID;

    ConcernRolePhoneNumberDtlsList concernRolePhoneNumberDtlsList =
      ConcernRolePhoneNumberFactory.newInstance().searchByConcernRole(concernRoleKey);

    ConcernRole concernRoleObj = ConcernRoleFactory.newInstance();
    ConcernRoleDtls concernRoleDtls = concernRoleObj.read(concernRoleKey);
    Date currentPrimaryPhoneNumberStartDate = Date.kZeroDate;

    List<SampleSortedPhoneNumber> list = new ArrayList<SampleSortedPhoneNumber>();

    for (ConcernRolePhoneNumberDtls concernRolePhoneNumberDtls:concernRolePhoneNumberDtlsList.dtls) {

      PhoneNumberKey phoneNumberKey = new PhoneNumberKey();
      phoneNumberKey.phoneNumberID = concernRolePhoneNumberDtls.phoneNumberID;

      if (concernRolePhoneNumberDtls.phoneNumberID == concernRoleDtls.primaryPhoneNumberID) {
        currentPrimaryPhoneNumberStartDate = concernRolePhoneNumberDtls.startDate;
      }

      SampleSortedPhoneNumber sampleSortedPhoneNumber =
        new SampleSortedPhoneNumber(concernRolePhoneNumberDtls);
      list.add(sampleSortedPhoneNumber);
    }

    Collections.sort(list);

    SampleSortedPhoneNumber newPrimaryPhoneNumber = list.get(0);

    if (newPrimaryPhoneNumber.getStartDate().after(currentPrimaryPhoneNumberStartDate)) {
      concernRoleDtls.primaryPhoneNumberID = newPrimaryPhoneNumber.getPhoneNumberID();
      concernRoleObj.pdcModify(concernRoleKey, concernRoleDtls);
    }
}
  class SampleSortedPhoneNumber implements Comparable<SampleSortedPhoneNumber> {
    private long phoneNumberID;
    private Date startDate;

    SampleSortedPhoneNumber(ConcernRolePhoneNumberDtls dtls) {
      this.phoneNumberID = dtls.phoneNumberID;
      this.startDate = dtls.startDate;
    }

    public long getPhoneNumberID() {
      return phoneNumberID;
    }

    public Date getStartDate() {
      return startDate;
    }
```

```
    public int compareTo(SampleSortedPhoneNumber o) {
      return o.getStartDate().compareTo(this.getStartDate());
    }
  }
}
```

5.4.3.2 Paso 2: Añadir un enlace a la nueva implementación de manejador principal

Se utilizan enlaces Guice para registrar la implementación.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Registrar la implementación de manejador principal
    bind(PDCPrimaryPhoneNumberHandler.class).to(
      SamplePrimaryPhoneNumberHandlerImpl.class);
  }
}
```

Nota: Los módulos Guice nuevos deben registrarse añadiendo una fila en la tabla de base de datos ModuleClassName. Consulte la publicación Persistence Cookbook para obtener más información.

5.5 Pruebas recíprocas

5.5.1 ¿Qué son pruebas recíprocas?

Las pruebas recíprocas son un tipo de pruebas que constan de dos pruebas que se deben procesar juntas. El tipo de pruebas de relación dinámicas es un ejemplo de pruebas recíprocas. Cuando la Persona A se registra como cónyuge de la la Persona B, se registran las pruebas de relación correspondientes, la Persona B es cónyuge de la Persona A. Cuando se insertan, modifican o eliminan pruebas para la Persona A, el sistema inserta, modifica o elimina las pruebas de relación correspondientes para la Persona B.

5.5.2 ¿Por qué es importante proporcionar una implementación de pruebas recíprocas?

Si desarrolla un nuevo tipo de pruebas dinámicas recíprocas, también debe proporcionar una implementación de la interfaz `ReciprocalEvidenceConversion`.

5.5.3 Implementaciones de pruebas recíprocas

Cuando se insertan, modifican o eliminan pruebas, se invoca un punto de gancho que, de forma predeterminada, desencadena la funcionalidad de manejador de pruebas recíprocas. Este punto de gancho de pruebas nuevo se denomina `GlobalEvidenceHook` y se puede encontrar en el paquete curam.core.sl.infrastructure.impl. La interfaz `GlobalEvidenceHook` permite que se produzca el proceso personalizado después de que se hayan completado las operaciones de prueba.

Interfaz GlobalEvidenceHook

La interfaz `GlobalEvidenceHook` contiene los métodos siguientes:

`postInsertEvidence` se invoca después que se hayan insertado pruebas y acepta dos parámetros:

- caseKey: el identificador del caso al que pertenecen las pruebas
- evKey: el identificador y tipo de las pruebas

`postModifyEvidence` se invoca después de que se hayan modificado las pruebas y acepta dos parámetros:

- caseKey: el identificador del caso al que pertenecen las pruebas
- evKey: el identificador y tipo de las pruebas

postRemoveEvidence se invoca después de que se hayan eliminado las pruebas y acepta dos parámetros:
- caseKey: el identificador del caso al que pertenecen las pruebas
- evKey: el identificador y tipo de las pruebas

postDiscardPendingUpdate se invoca después de que se descarte una actualización pendiente de las pruebas y acepta dos parámetros:
- caseKey: el identificador del caso al que pertenecen las pruebas
- evKey: el identificador y tipo de las pruebas

postDiscardPendingRemove se invoca después de que se descarte una eliminación pendiente de las pruebas y acepta dos parámetros:
- caseKey: el identificador del caso al que pertenecen las pruebas
- evKey: el identificador y tipo de las pruebas

Manejador de pruebas recíprocas

La implementación predeterminada para GlobalEvidenceHook invoca la funcionalidad de manejador de pruebas recíprocas. El manejador de pruebas recíprocas es responsable de todo el proceso de pruebas recíprocas común. Localiza las pruebas recíprocas y, si se encuentran, realiza en ellas los mismos cambios que se han realizado en las pruebas originales. Si no se encuentran las pruebas recíprocas y se han insertado las pruebas originales, insertará las pruebas recíprocas correspondiente. Dado que el manejador de pruebas recíprocas es básico para el proceso de pruebas recíprocas, no se puede personalizar directamente, pero se puede personalizar por medio de GlobalEvidenceHook, si fuera necesario.

Interfaz de conversión de pruebas recíprocas

La interfaz ReciprocalEvidenceConversion es responsable de la comparación de pruebas recíprocas y originales, de la recuperación de participantes y de crear pruebas recíprocas nuevas y modificadas a partir de las pruebas originales. Para que las pruebas personalizadas sean recíprocas, se debe proporcionar una implementación de interfaz ReciprocalEvidenceConversion. Mientras que el manejador no es consciente de la estructura interna de las pruebas, la implementación de interfaz de conversión sí lo es, de modo que es aquí donde se encuentra el principal punto de personalización. La interfaz ReciprocalEvidenceConversion se puede encontrar en el paquete curam.core.sl.infrastructure.impl y contiene los métodos siguientes:
- Object getReciprocal(final Object original, final long targetCaseID): Crea detalles de pruebas recíprocas a partir de los detalles de pruebas originales
- Object getUpdatedReciprocal(final Object original, final Object unmodifiedReciprocal): Crea detalles de pruebas recíprocas modificadas a partir de los detalles de pruebas originales y de los detalles de pruebas recíprocas no modificadas
- long getPrimaryParticipant(final Object originalEvidence): Recupera el participante principal (ID de rol de asunto) de las pruebas originales. Tenga en cuenta que el participante principal de las pruebas originales es el participante relacionado en las pruebas recíprocas
- long getRelatedParticipant(final Object originalEvidence): Recupera el participante relacionado (ID de rol de asunto) de las pruebas originales. Tenga en cuenta que el participante relacionado de las pruebas originales es el participante principal en las pruebas recíprocas
- boolean matchEvidenceDetails(final Object evidenceDetails1, final Object evidenceDetails2): Comprueba los detalles de pruebas para encontrar una coincidencia. La implementación de este método determina si dos detalles de pruebas pasados se consideran como una coincidencia.

El apartado siguiente muestra cómo implementar pruebas recíprocas.

5.5.4 Ejemplo de implementación de pruebas recíprocas

El ejemplo siguiente muestra cómo implementar pruebas recíprocas. En este escenario, se ha configurado la relación laborable como un nuevo tipo de pruebas dinámicas. Para obtener más información sobre cómo configurar un nuevo tipo de pruebas, consulte la Guía de configuración de pruebas dinámicas de Cúram. El nuevo tipo de pruebas de relación laborable se ha identificado como recíproco y tiene los atributos siguientes:

- participant: ID de rol de participante en el caso de la persona/candidato para la que se están entrando las pruebas
- relatedParticipant: ID de rol de participante de caso de la persona/candidato relacionado
- workingRelationship: relación de trabajo entre los dos participantes

Se ha supuesto que este tipo de pruebas dinámicas se ha activado y está configurado para utilizarse con personas o candidatos. Para que la infraestructura maneje estas pruebas recíprocas correctamente, se debe proporcionar una implementación de la interfaz `ReciprocalEvidenceConversion`.

Los pasos implicados son:

- Proporcionar una implementación de conversión de pruebas recíprocas.
- Añadir un enlace a la nueva implementación de conversión de pruebas recíprocas.

5.5.4.1 Paso 1: Proporcionar una implementación de conversión de pruebas recíprocas

```
public class SampleWorkingRelationshipReciprocalConversion
  implements ReciprocalEvidenceConversion {

  @Inject
  private EvidenceTypeDefDAO etDefDAO;

  @Inject
  private EvidenceTypeVersionDefDAO etVerDefDAO;

  public SampleWorkingRelationshipReciprocalConversion() {

  }

  public Object getReciprocal(
    final Object original, final long targetCaseID)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails = (DynamicEvidenceDataDetails) original;

    String workingRelationshipOriginal =
      originalDetails.getAttribute("workingRelationship").getValue();

    String workingRelationshipRec = "";

    if (workingRelationshipOriginal.equals("ISMANAGEROF")) {
      workingRelationshipRec = "ISMANAGEDBY";
    }

    EvidenceTypeKey evdType = new EvidenceTypeKey();
    evdType.evidenceType = "WORKINGRELATIONSHIP";

    EvidenceTypeDef evdTypeDef =
      etDefDAO.readActiveEvidenceTypeDefByTypeCode(evdType.evidenceType);

    EvidenceTypeVersionDef evTypeVersion =
      etVerDefDAO.getActiveEvidenceTypeVersionAtDate(evdTypeDef, Date.getCurrentDate());

    DynamicEvidenceDataDetails reciprocalDetails =
      DynamicEvidenceDataDetailsFactory.newInstance(evTypeVersion);
```

```
  DynamicEvidenceDataAttributeDetails workingRelationshipAttr =
    reciprocalDetails.getAttribute("workingRelationship");

  DynamicEvidenceTypeConverter.setAttribute(workingRelationshipAttr, workingRelationshipRec);

  DynamicEvidenceDataAttributeDetails participantAttr =
    reciprocalDetails.getAttribute("participant");

  DynamicEvidenceTypeConverter.setAttribute(participantAttr,
      originalDetails.getAttribute("relatedParticipant").getValue());

  DynamicEvidenceDataAttributeDetails relatedParticipantAttr =
    reciprocalDetails.getAttribute("relatedParticipant");

  DynamicEvidenceTypeConverter.setAttribute(relatedParticipantAttr,
      originalDetails.getAttribute("participant").getValue());

  return reciprocalDetails;
}

public Object getUpdatedReciprocal(
  final Object original, final Object unmodifiedReciprocal)
  throws AppException, InformationalException {

  DynamicEvidenceDataDetails originalDetails =
    (DynamicEvidenceDataDetails) original;
  DynamicEvidenceDataDetails reciprocalDetails =
    (DynamicEvidenceDataDetails) unmodifiedReciprocal;

  long caseParticipantRoleIDRec = Long.parseLong(
    reciprocalDetails.getAttribute("participant").getValue());
  long relCaseParticipantRoleIDRec = Long.parseLong(
    reciprocalDetails.getAttribute("relatedParticipant").getValue());
  String workingRelationshipRec = reciprocalDetails.getAttribute("workingRelationship").getValue();

  for (final DynamicEvidenceDataAttributeDetails listDetails: originalDetails.getAttributes()) {
    reciprocalDetails.getAttribute(listDetails.getName()).setValue(
      listDetails.getValue());
  }

  DynamicEvidenceDataAttributeDetails workingRelationshipAttr =
    reciprocalDetails.getAttribute("workingRelationship");

  DynamicEvidenceTypeConverter.setAttribute(workingRelationshipAttr, workingRelationshipRec);

  DynamicEvidenceDataAttributeDetails participantAttr =
    reciprocalDetails.getAttribute("participant");

  DynamicEvidenceTypeConverter.setAttribute(participantAttr,
    caseParticipantRoleIDRec);

  DynamicEvidenceDataAttributeDetails relatedParticipantAttr =
    reciprocalDetails.getAttribute("relatedParticipant");

  DynamicEvidenceTypeConverter.setAttribute(relatedParticipantAttr,
    relCaseParticipantRoleIDRec);

  return reciprocalDetails;
}

public boolean matchEvidenceDetails(
  final Object evidenceDetails1, final Object evidenceDetails2)
  throws AppException, InformationalException {

  DynamicEvidenceDataDetails dynamicEvidenceDataDetails1 =
    (DynamicEvidenceDataDetails) evidenceDetails1;
  DynamicEvidenceDataDetails dynamicEvidenceDataDetails2 =
```

```
    (DynamicEvidenceDataDetails) evidenceDetails2;

  curam.core.sl.intf.CaseParticipantRole caseParticipantRoleObj =
    curam.core.sl.fact.CaseParticipantRoleFactory.newInstance();

  CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();
  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails1.getAttribute("participant"));

  Long concernRoleID1 =
    caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails1.getAttribute("relatedParticipant"));

  Long relConcernRoleID1 =
    caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails2.getAttribute("participant"));

  Long concernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails2.getAttribute("relatedParticipant"));

  Long relConcernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

  return dynamicEvidenceDataDetails1.getAttribute("workingRelationship").getValue().equals(
    dynamicEvidenceDataDetails2.getAttribute("workingRelationship").getValue())
    && (concernRoleID1.longValue() == concernRoleID2.longValue())
    && (relConcernRoleID1.longValue() == relConcernRoleID2.longValue());
}

public boolean matchOriginalAndReciprocal(
  final Object originalEvidence, final Object reciprocalEvidence)
  throws AppException, InformationalException {

  DynamicEvidenceDataDetails originalDetails =
    (DynamicEvidenceDataDetails) originalEvidence;

  DynamicEvidenceDataDetails reciprocalDetails =
    (DynamicEvidenceDataDetails) reciprocalEvidence;

  curam.core.sl.intf.CaseParticipantRole caseParticipantRoleObj =
    curam.core.sl.fact.CaseParticipantRoleFactory.newInstance();
  CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();

  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    originalDetails.getAttribute("participant"));

  Long concernRoleID1 = caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    originalDetails.getAttribute("relatedParticipant"));

  Long relConcernRoleID1 = caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    reciprocalDetails.getAttribute("participant"));

  Long concernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

  caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
    reciprocalDetails.getAttribute("relatedParticipant"));

  Long relConcernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;
```

```
    String workingRelationshipOriginal =
      originalDetails.getAttribute("workingRelationship").getValue();

    String workingRelationshipRec = "";
    if (workingRelationshipOriginal.equals("ISMANAGEROF")) {
      workingRelationshipRec = "ISMANAGEDBY";
    }

    return reciprocalDetails.getAttribute("workingRelationship").getValue().equals(
      workingRelationshipRec)
      && (concernRoleID1.longValue() == relConcernRoleID2.longValue())
      && (relConcernRoleID1.longValue() == concernRoleID2.longValue());
  }

  public long getPrimaryParticipant(final Object originalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails = (DynamicEvidenceDataDetails) originalEvidence;

    long caseParticipantRoleID = Long.parseLong(
      originalDetails.getAttribute("participant").getValue());

    CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID = caseParticipantRoleID;

    return CaseParticipantRoleFactory.newInstance().read(caseParticipantRoleKey).participantRoleID;
  }

  public long getRelatedParticipant(final Object originalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
      (DynamicEvidenceDataDetails) originalEvidence;

    long caseParticipantRoleID = Long.parseLong(
      originalDetails.getAttribute("relatedParticipant").getValue());

    CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID = caseParticipantRoleID;

    return CaseParticipantRoleFactory.newInstance().read(caseParticipantRoleKey).participantRoleID;
  }
}
```

5.5.4.2 Paso 2: Añadir un enlace a la nueva implementación de conversión de pruebas recíprocas

Se utilizan enlaces Guice para registrar la implementación.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    MapBinder<CASEEVIDENCEEntry, ReciprocalEvidenceConversion> recEvidenceConversionMapBinder =
      MapBinder.newMapBinder(binder(), CASEEVIDENCEEntry.class, ReciprocalEvidenceConversion.class);

    reciprocalEvidenceConversionMapBinder.addBinding(CASEEVIDENCEEntry.get("WORKINGRELATIONSHIP")).to(
      SampleWorkingRelationshipReciprocalConversion.class);
  }
}
```

Nota: Los módulos Guice nuevos deben registrarse añadiendo una fila en la tabla de base de datos ModuleClassName. Consulte la publicación Persistence Cookbook para obtener más información.

5.5.5 Limitaciones de pruebas recíprocas

La infraestructura de manejo de pruebas recíprocas tiene las limitaciones siguientes:

- Las pruebas deben ser pruebas temporales. Puede ser pruebas estáticas, dinámicas o generadas.
- Las pruebas deben tener un participante y un participante relacionado, alternativamente, el código de implementación de `ReciprocalEvidenceConversion` debe ser capaz de determinar el participante y el participante relacionado utilizando los detalles de pruebas.
- Cuando las pruebas recíprocas y las pruebas originales relacionadas están en el mismo caso, los cambios deben aplicarse siempre juntos porque, de lo contrario, los datos de pruebas originales y recíprocas estarán fuera de sincronización.
- Las pruebas recíprocas se pueden procesar automáticamente sólo si ambos participantes relacionados están registrados como participantes MEMBER o PRIMARY en el mismo caso o las pruebas se han registrado como pruebas de persona/candidato.

5.6 Propietario de caso de datos de participante

5.6.1 ¿Por qué cambiar el propietario de caso de datos de participante?

Puede que sea necesario cambiar el propietario de caso de datos de participante si se necesita más control en torno a la propiedad de los participantes.

5.6.2 Cambio del propietario de caso de datos de participante

Cuando se registra una persona o un candidato en el sistema, se crea un caso en el fondo para ayudar a gestionar estos datos. Esto se conoce también como "caso de datos de participante'. De forma predeterminada, este caso tiene un propietario de caso, el usuario que ha iniciado la sesión. Es posible cambiarlo por un propietario de caso diferentes mediante la interfaz `PDCCaseOwnerAssignmentStrategy`. La interfaz `PDCCaseOwnerAssignmentStrategy` se puede encontrar en el paquete curam.pdc.impl y tiene un método `createOwner`. Acepta dos parámetros:

- key: identificador del caso de datos de participante
- ownerDtls: detalles del propietario de caso de datos de participante

5.6.3 Cambio del ejemplo de propietario de caso de datos de participante

El ejemplo siguiente describe cómo cambiar el propietario de caso de datos de participante; en este escenario el propietario se establece en el usuario del sistema.

Los pasos implicados son:

- Proporcionar una implementación de la estrategia de asignación de propietario de un caso que establecerá el propietario del caso.
- Añadir un enlace a la implementación de la estrategia de asignación de propietario de caso.

5.6.3.1 Paso 1: Proporcionar una implementación de estrategia de asignación de propietario de caso

El fragmento de código siguiente muestra una implementación de ejemplo para `PDCCaseOwnerAssignmentStrategy`; simplemente establece que el propietario sea el usuario del sistema.

```
@Singleton
public class SampleCaseOwnerAssignmentStrategyImpl
  implements PDCCaseOwnerAssignmentStrategy {

  public void createOwner(CaseHeaderKey key, OrgObjectLinkDtls ownerDtls)
    throws AppException, InformationalException {

    ownerDtls.orgObjectType = ORGOBJECTTYPE.USER;
```

```
    ownerDtls.userName = UserAccessFactory.newInstance().getSystemUserDetails().userName;

    OrgObjectLinkFactory.newInstance().insert(ownerDtls);

    OrgObjectLinkKey orgObjectLinkKey = new OrgObjectLinkKey();
    orgObjectLinkKey.orgObjectLinkID = ownerDtls.orgObjectLinkID;

    CaseUserRoleDtls caseUserRoleDtls = new CaseUserRoleDtls();
    caseUserRoleDtls.caseID = key.caseID;
    caseUserRoleDtls.orgObjectLinkID = orgObjectLinkKey.orgObjectLinkID;
    caseUserRoleDtls.typeCode = CASEUSERROLETYPE.OWNER;
    caseUserRoleDtls.recordStatus = RECORDSTATUS.NORMAL;

    curam.core.sl.entity.fact.CaseUserRoleFactory.newInstance().insert(caseUserRoleDtls);

    CaseHeader caseHeaderObj = CaseHeaderFactory.newInstance();
    CaseHeaderDtls caseHeaderDtls = caseHeaderObj.read(key);
    caseHeaderDtls.ownerOrgObjectLinkID = orgObjectLinkKey.orgObjectLinkID;
    caseHeaderObj.modify(key, caseHeaderDtls);
  }
}
```

5.6.3.2 Paso 2: Añadir un enlace a la implementación de estrategia de asignación de propietario de caso

Se utilizan enlaces Guice para registrar la implementación.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Registrar la implementación
    bind(PDCCaseOwnerAssignmentStrategy.class)
   .to(SampleCaseOwnerAssignmentStrategyImpl.class);
  }
}
```

Nota: Los módulos Guice nuevos deben registrarse añadiendo una fila en la tabla de base de datos ModuleClassName. Consulte la publicación Persistence Cookbook para obtener más información.

Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en los Estados Unidos. Es posible que IBM no ofrezca los productos, servicios o características que se describen en este documento en otros países. Solicite información al representante local de IBM acerca de los productos y servicios disponibles actualmente en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implica que sólo pueda utilizarse ese producto, programa o servicio de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ningún derecho de propiedad intelectual de IBM. No obstante, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM. IBM podría tener patentes o solicitudes de patentes pendientes relacionadas con el tema principal que se describe en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede enviar consultas sobre licencias, por escrito, a:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

EE.UU.

Para consultas sobre licencias relacionadas con información de doble byte (DBCS), póngase en contacto con el departamento de propiedad intelectual de IBM de su país o envíe sus consultas, por escrito, a:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokio 103-8510, Japón

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país donde las disposiciones en él expuestas sean incompatibles con la legislación local: INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN GARANTÍA DE NINGUNA CLASE, YA SEA EXPLÍCITA O IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERABILIDAD, COMERCIALIZACIÓN O IDONEIDAD PARA UN PROPÓSITO DETERMINADO. Algunos países no permiten la renuncia a garantías explícitas o implícitas en determinadas transacciones, por lo que puede que esta declaración no sea aplicable en su caso.

La información de este documento puede incluir imprecisiones técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; estos cambios se incorporarán en nuevas ediciones de la publicación. IBM puede reservarse el derecho de realizar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento sin previo aviso.

Cualquier referencia incluida en esta información a sitios web que no sean de IBM sólo se proporciona para su comodidad y en ningún modo constituye una aprobación de dichos sitios web. El material de esos sitios web no forma parte del material de este producto de IBM y la utilización de esos sitios web se realizará bajo su total responsabilidad.

IBM puede utilizar o distribuir cualquier información que se le facilite de la manera que considere adecuada, sin contraer por ello ninguna obligación con el remitente. Los titulares de licencias de este programa que deseen obtener información sobre el mismo con el fin de permitir: (i) el intercambio de información entre programas creados independientemente y otros programas (incluido éste) y el uso mutuo de información que se haya intercambiado, deben ponerse en contacto con:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

EE.UU.

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una cuota.

IBM proporciona el programa bajo licencia que se describe en este documento y todo el material bajo licencia disponible para el mismo bajo los términos del Acuerdo de cliente de IBM, el Acuerdo internacional de licencias de programas de IBM o cualquier acuerdo equivalente entre las partes.

Los datos de rendimiento incluidos aquí se determinaron en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar considerablemente. Algunas mediciones podrían haberse realizado en sistemas en desarrollo y, por lo tanto, no existe ningún tipo de garantía de que dichas mediciones sean las mismas en los sistemas con disponibilidad general. Además, es posible que algunas mediciones se hayan calculado mediante extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables a sus entornos específicos.

La información relacionada con productos que no son de IBM se ha obtenido de los proveedores de dichos productos, de sus anuncios publicados o de otras fuentes de disponibilidad pública.

IBM no ha probado estos productos y no puede confirmar la precisión de rendimiento, compatibilidad ni otras afirmaciones relacionadas con productos que no son de IBM. Las preguntas relativas a las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de dichos productos.

Las afirmaciones relativas a las intenciones futuras de IBM están sujetas a cambio o retirada sin previo aviso, y sólo representan objetivos

Todos los precios de IBM que se muestran son precios de distribuidor recomendados por IBM, corresponden al momento actual y están sujetos a cambios sin aviso previo. Los precios de los distribuidores pueden variar.

Esta información se ofrece con fines de planificación únicamente. La información incluida en este documento puede cambiar antes de que los productos descritos estén disponibles.

Esta información contiene ejemplos de datos e informes utilizados en operaciones comerciales diarias. Para ilustrarlos de la manera más completa posible, los ejemplos incluyen los nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier parecido con nombres y direcciones utilizados por empresas comerciales reales son mera coincidencia.

LICENCIA DE COPYRIGHT:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente, que ilustran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir los programas de ejemplo de cualquier forma, sin tener que pagar a IBM, con intención de desarrollar, utilizar, comercializar o distribuir programas de aplicación que estén en conformidad con la interfaz de programación de aplicaciones (API) de la plataforma operativa para la que están escritos los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por lo tanto, IBM no puede garantizar ni implicar la fiabilidad, capacidad de servicio o función de estos programas. Los programas de ejemplo se proporcionan "TAL CUAL", sin garantía de ningún tipo. IBM no es responsable de ningún daño resultante de la utilización de los programas de ejemplo por parte del usuario.

Todas las copias o fragmentos de las copias de estos programas de ejemplo o cualquier trabajo que de ellos se derive, deberán incluir un aviso de copyright como el que se indica a continuación:

© (el nombre de la empresa) (año). Algunas partes de este código proceden de los programas de ejemplo de IBM Corp.

© Copyright IBM Corp. _escriba el año o los años_. Reservados todos los derechos.

Si visualiza esta información en una copia software, es posible que no aparezcan las fotografías ni las ilustraciones en color.

Marcas registradas

IBM, el logotipo de IBM e ibm.com son marcas registradas de International Business Machines Corp., registradas en muchas jurisdicciones en todo el mundo. Otros nombres de productos y servicios pueden ser marcas registradas de IBM u otras empresas. Encontrará una lista actual de marcas registradas de IBM en la web en "Copyright and trademark information" en http://www.ibm.com/legal/us/en/copytrade.shtml.

Otros nombres pueden ser marcas registradas de sus respectivos propietarios. Otros nombres de empresas, productos o servicios pueden ser marcas registradas o de servicio de terceros.

Impreso en España