Professional Edition v 0.1.x

Tutorial

Revisión: 03/10/17

Sumario

1 Introducción

Este manual muestra cómo utilizar Samsara Software Professional Edition para modelar desde el principio hasta el final una aplicación J2EE con base de datos Oracle, para la gestión de incidencias.

2 Software adicional

Los requisitos software son los siguientes:

Oracle XE 11g_2:

1. Instalar el servidor de base de datos.
2. Ir a la URL http://127.0.0.1:8080/apex/f?p=4950 y crear un usuario de base de datos DB_USER desde la pestaña "Application Express".

JDK 1.6_45: Instalar el JDK 6_45, no es necesario ni el código fuente ni la JRE pública.

JBoss GA 5.1:

1. Descargar el zip y descomprimirlo.
2. Copiar jboss-5.1.0.GA\server\default a jboss-5.1.0.GA\jboss01
3. Modificar el fichero jboss-5.1.0.GA\server\jboss01\conf\bindingservice.beans\META-INF\bindings-jboss-beans.xml

Modificar el siguiente código:

```
<!-- Provides management tools with a ProfileService ManagementView
    interface to the SBM and its components -->
<bean name="ServiceBindingManagementObject"
    class="org.jboss.services.binding.managed.ServiceBindingManagementObject">

  <constructor>
    <!-- The name of the set of bindings to use for this server -->
    <parameter>${jboss.service.binding.set:ports-default}</parameter>

    <!--  The binding sets -->
    <parameter>
      <set>
        <inject bean="PortsDefaultBindings"/>
        <inject bean="Ports01Bindings"/>
        <inject bean="Ports02Bindings"/>
        <inject bean="Ports03Bindings"/>
      </set>
    </parameter>

    <!-- Base binding metadata that is used to create bindings for each set -->
    <parameter><inject bean="StandardBindings"/></parameter>

  </constructor>
</bean>
```

Por:

```
<bean name="ServiceBindingManagementObject"
    class="org.jboss.services.binding.managed.ServiceBindingManagementObject">

  <constructor>
    <!-- The name of the set of bindings to use for this server -->
    <parameter>${jboss.service.binding.set:jboss01}</parameter>

    <!--  The binding sets -->
    <parameter>
      <set>
          <inject bean="Jboss01Bindings"/>
      </set>
    </parameter>

    <!-- Base binding metadata that is used to create bindings for each set -->
    <parameter><inject bean="StandardBindings"/></parameter>

  </constructor>
</bean>

 <bean name="Jboss01Bindings" class="org.jboss.services.binding.impl.ServiceBindingSet">
  <constructor>
    <!--  The name of the set -->
    <parameter>jboss01</parameter>
    <!-- Default host name -->
    <parameter>${jboss.bind.address}</parameter>
    <!-- The port offset -->
    <parameter>20000</parameter>
    <!-- Set of bindings to which the "offset by X" approach can't be applied -->
    <parameter><null/></parameter>
  </constructor>
</bean>
```

4. crear un fichero jboss-5.1.0.GA\bin\run_jboss01.bat con el siguiente contenido:

```
set "JAVA_OPTS="
rem # Reduce the RMI GCs to once per hour for Sun JVMs.
set "JAVA_OPTS=%JAVA_OPTS% -Dsun.rmi.dgc.client.gcInterval=3600000
-Dsun.rmi.dgc.server.gcInterval=3600000"
rem # Warn when resolving remote XML DTDs or schemas.
set "JAVA_OPTS=%JAVA_OPTS% -Dorg.jboss.resolver.warning=true"
set "JAVA_OPTS=%JAVA_OPTS% -Xmx2048m -XX:MaxPermSize=1g
-Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n"

set JBOSS_HOME=[ruta al directorio de instalación]\jboss-5.1.0.GA
set JAVA_HOME=[ruta al directorio de instalación]a\jdk1.6.0_45
run.bat -c jboss01
```

5. añadir al fichero el fichero jboss-5.1.0.GA\server\jboss01\conf\login-config.xml

```
<policy>
 <!-- Used by clients within the application server VM such as
 mbeans and servlets that access EJBs.
 -->

  <application-policy name="seycon">
    <authentication>
      <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule" flag="required">
        <module-option name="dsJndiName">java:com.samsarasoftware.base_project.db</module-option>
        <module-option name="principalsQuery">select password from SUser  where name=?</module-option>
        <module-option name="rolesQuery">select 'INC_USER', 'Roles' from suser where name=?</module-
option>
                        <module-option name="hashStorePassword">false</module-option>
                        <module-option name="hashUserPassword">false</module-option>
      </login-module>
    </authentication>
  </application-policy>
```

Con esto ya podremos ejecutar nuestro servidor de aplicaciones J2EE.

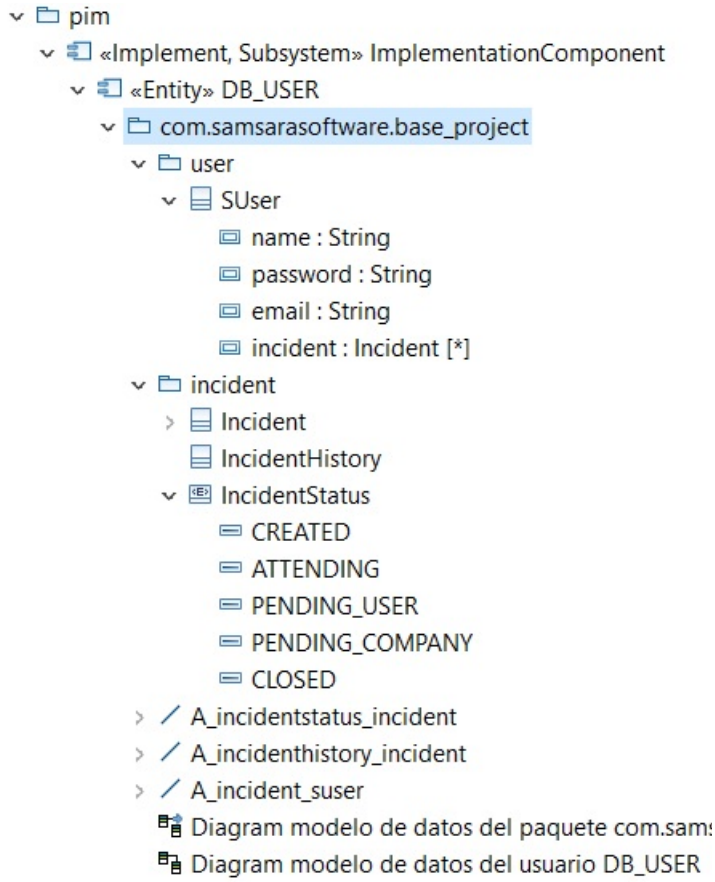
3 Analizando el sistema

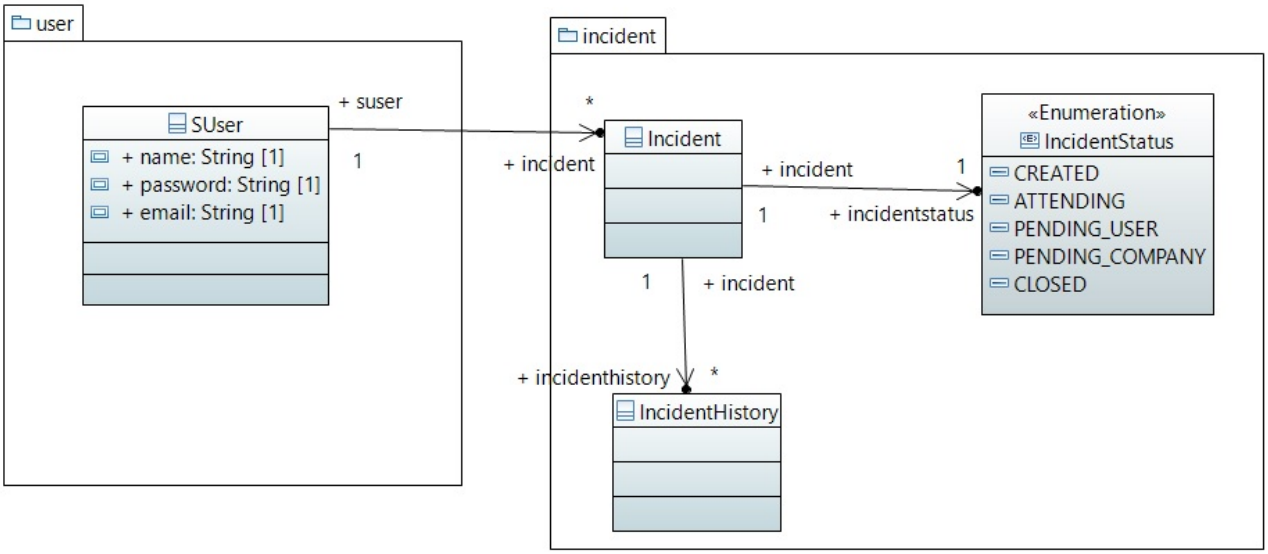
Lo primero que debemos hacer es descargar el proyecto base, que ya contiene un proyecto estructurado, tal como se explica en el manual de primeros pasos. Abrimos el documento *src/main/uml/proyecto_base.di* con Papyrus, y ya podemos empezar a modelar en UML. Podemos encontrar el código fuente del proyecto terminado en este enlace.

3.1 Definición del modelo de datos del sistema

Abrimos la vista "Model explorer" de Papyrus, y abrimos la siguiente ruta en el documento UML: *pim / <<Implement, Subsystem>> ImplementationComponent / <<Entity>>DB_USER / com.samsarasoftware.base_project*

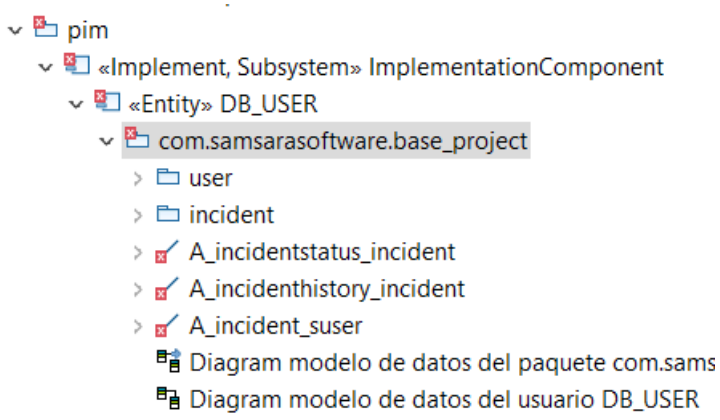
En este paquete, vemos que hay un diagrama de clases ya creado con nombre "modelo de datos del paquete com.samsarasoftware.base_project". Lo abrimos, y creamos la siguiente estructura de clases:


pim
«Implement, Subsystem» ImplementationComponent
«Entity» DB_USER
com.samsarasoftware.base_project
user
SUser
name : String
password : String
email : String
incident : Incident [*]
incident
Incident
IncidentHistory
IncidentStatus
CREATED
ATTENDING
PENDING_USER
PENDING_COMPANY
CLOSED
A_incidentstatus_incident
A_incidenthistory_incident
A_incident_suser
Diagram modelo de datos del paquete com.sam
Diagram modelo de datos del usuario DB_USER

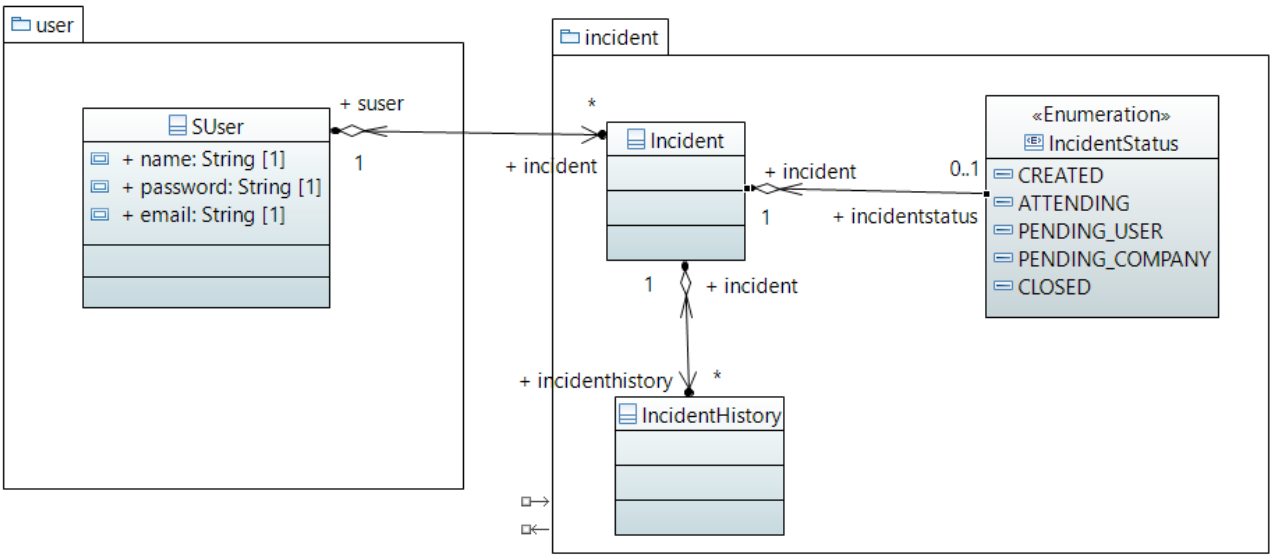

user
SUser
+ name: String [1]
+ password: String [1]
+ email: String [1]
+ suser
1
incident
*
+ incident
Incident
+ incident
1
1
+ incidentstatus
«Enumeration»
IncidentStatus
CREATED
ATTENDING
PENDING_USER
PENDING_COMPANY
CLOSED
1
+ incident
+ incidenthistory
*
IncidentHistory

3.2 Validación del modelo de datos del sistema

Abrimos el documento *src/main/uml/proyecto_base.uml* con el "UML Model Editor", y siguiendo los pasos explicados en el [módulo eclipse de Samsara Sfotware Comunity Edition] validamos el modelo.



Vemos que hay errores en el modelo, y siguiendo las instrucciones dadas al colocar el ratón encima del icono de error, corregimos el modelo de la siguiente forma:



3.3 Creación de las claves primarias

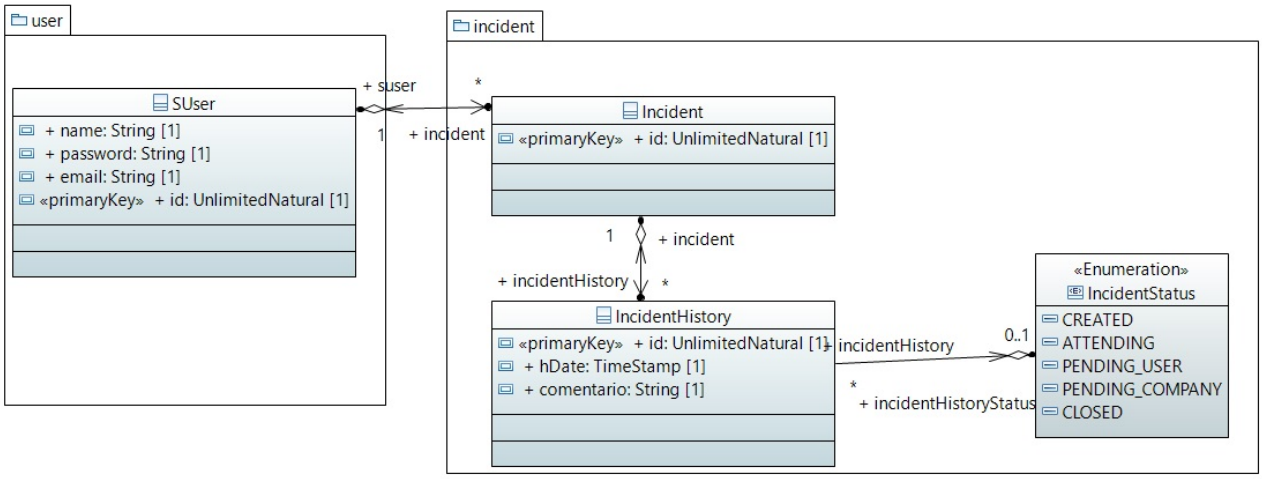
Todas las tablas de base de datos deben tener una clave primaria, para poder relacionarlas, o para buscar elementos en ella. Una clave primaria es un atributo de la clase, con el estereotipo <<database::primaryKey>>. En caso que utilicemos herencia de clases, y dependiendo del tipo de herencia, las sub-clases necesitan, o no, clave primaria.

Para agilizar el proceso de creación de claves primarias de forma correcta, disponemos de la transformación *generators / pim / adPrimaryKeyToEntities*. Vamos a ejecutarla y ver su resultado. Para ejecutarla, debemos ejecutar el comando *mvn clean install* en el directorio *generators / pim / adPrimaryKeyToEntities*.

Si hay algun problema ejecutando la transformación, debemos revisar que las rutas utlizadas por los generadores son correctas, en el fichero *generators/pom.xml* Si las modificamos, debemos ejecutar el comando *mvn clean install* en el directorio generators, y luego volver a ejecutar la transformación.

3.4 Ajuste del modelo de datos

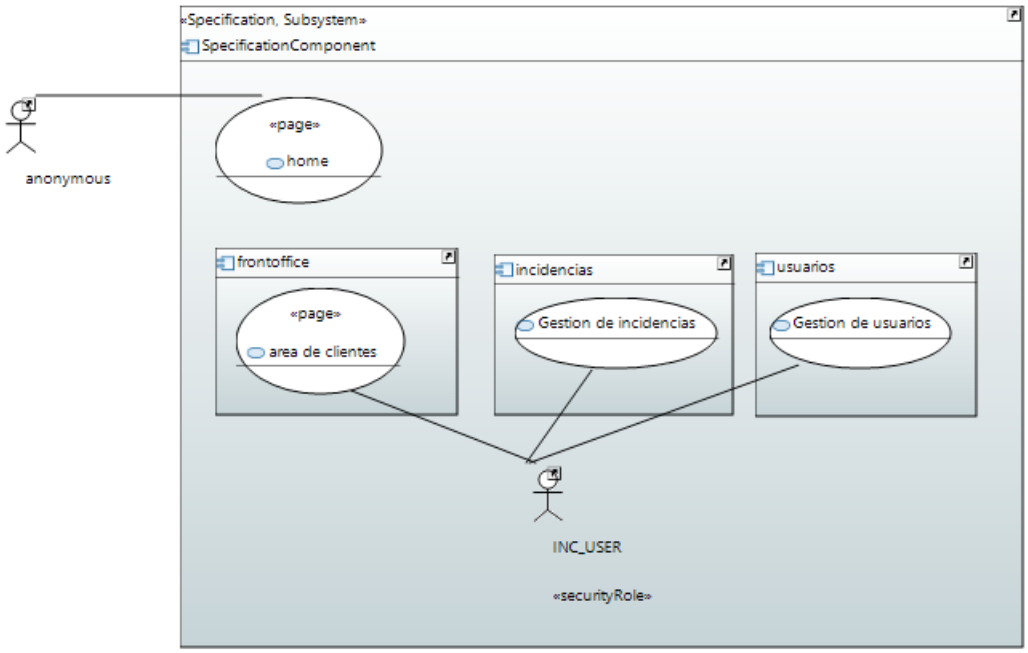
Finalmente, tras ver varios errores en el modelo, hacemos las últimas modificaciones:

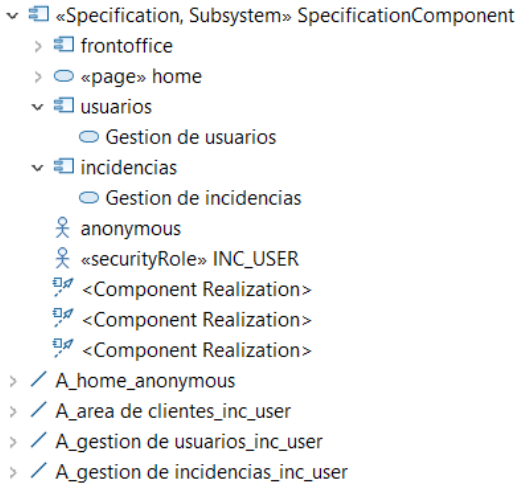


3.5 Definición de los componentes del sistema y los casos de uso

Abrimos la vista "Model explorer" de Papyrus, y abrimos la siguiente ruta en el documento UML: *cim / <<Specification, Subsystem>> SpecificationComponent*.

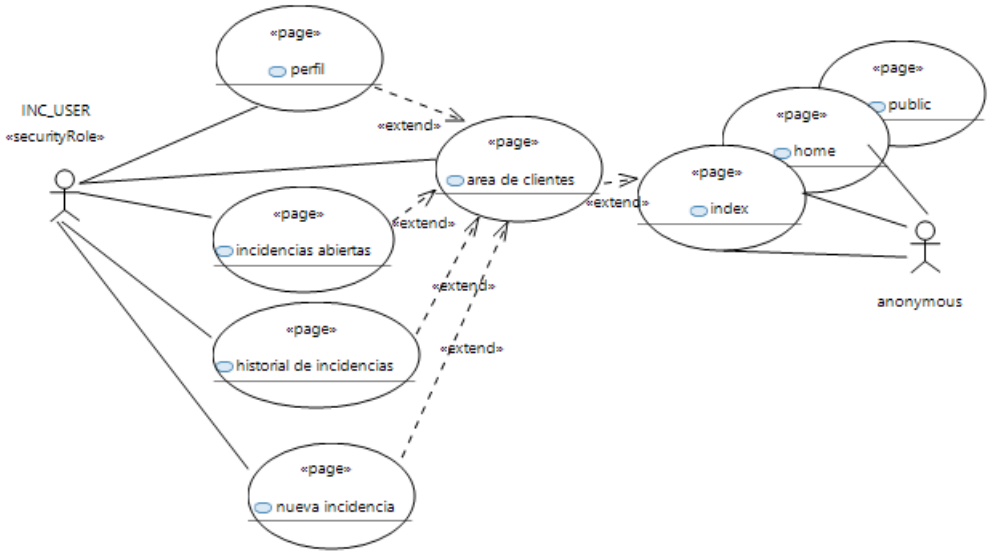
Vamos a crear cuatro componentes, un front-office para los usuarios, y otro backoffice para la empresa, u componente con los servicios de usuarios, y otro componente con los servicios de incidencias. INC_USER será el rol de usuario del cliente que puede ver y crear incidencias.





Las funcionalidades para los usuarios van a ser:

1. Gestión de usuarios
    1. Ver su información personal, y poder modificarla
2. Gestión de incidencias:
    1. ver sus incidencias abiertas, y comentarlas o cerrarlas
    2. crear incidencias
    3. ver el historial de todas las incidencias.



3.5.1 página inicial

Explicación:

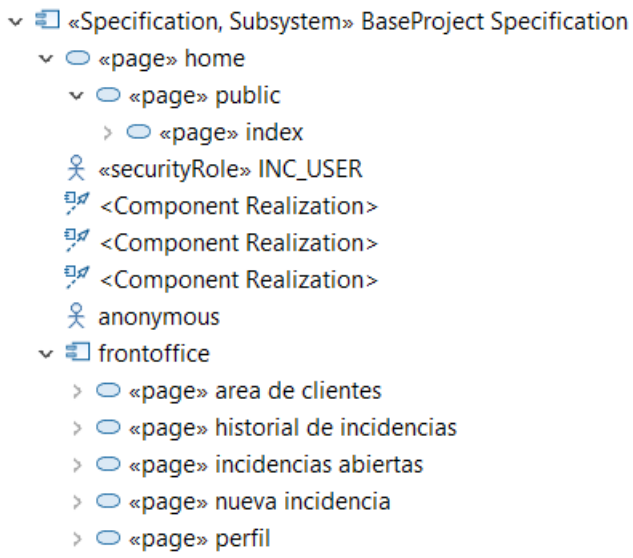
Creamos una estructura de directorios mediante páginas: home / public / index

Creamos un enlace de la página de inicio (index) a la página de área de clientes.

La página de área de clientes, tiene tres enlaces:

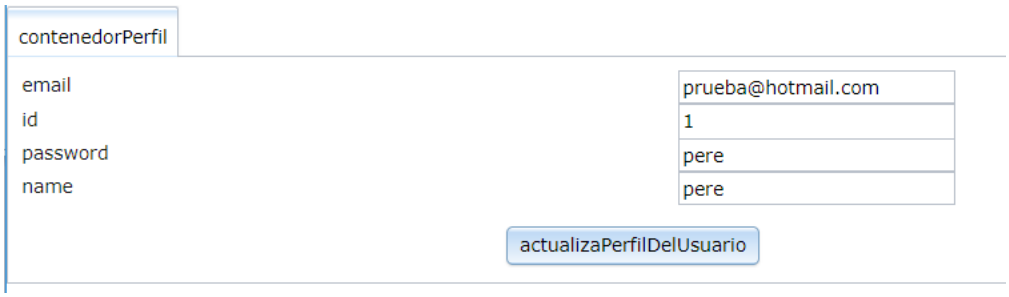
- A la página de perfil de usuario
- A la página de incidencias abiertas del usuario
- A la página de historial de incidencias del usuario

La distribución de los casos de uso en componentes es la siguiente:

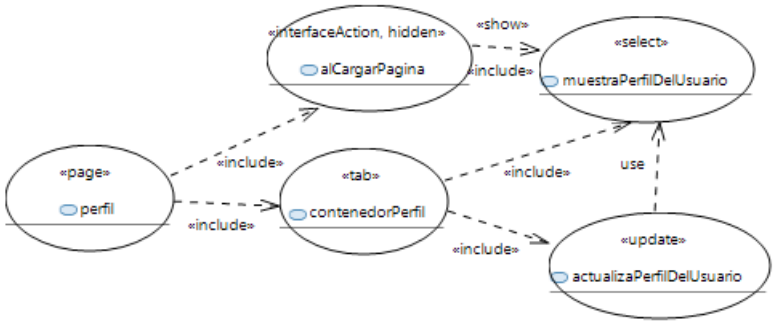


3.5.2 perfil de usuario

Para que quede más claro, veamos qué queremos obtener:



La página de perfil de usuario, tendrá un formulario de consulta y modificación de los datos del usuario. Una acción de interfaz cargará los datos al cargar la página.



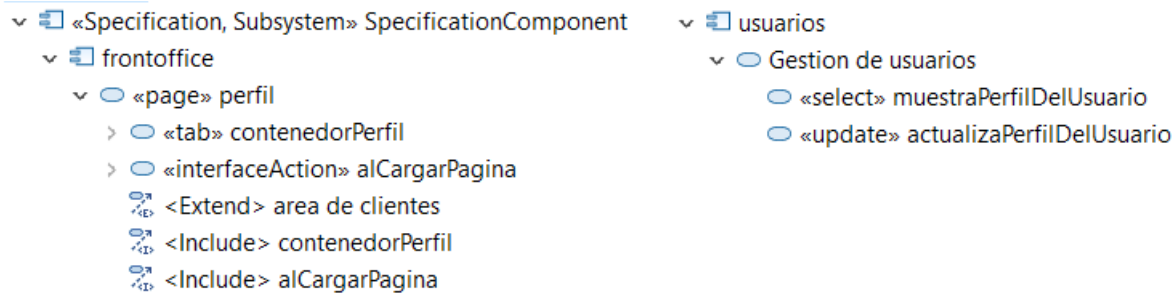
Explicación:

Creamos una página "perfil", que contiene una pestaña "contenedorPerfil". Este contenedor contiene un formulario para la edición del perfil del usuario. El formulario de edición utiliza, para cargar sus datos, una función de consulta, que debe estar incluída en la página. No se genera el formulario de consulta, ya que hay un formulario de edición que depende de éste, pero sí se crean las funciones necesarias para su ejecución.

También definimos una acción a realizar al cargarse la página. Le aplicamos el estereotipo <<interface::hidden>> para que no se muestre el botón de invocación a la función en la página. Al ejecutarse, invocará el servicio de consulta del perfil del usuario, y mostrará los resultados (al no haber un formulario de consulta, se muestra en el formulario de actualización)

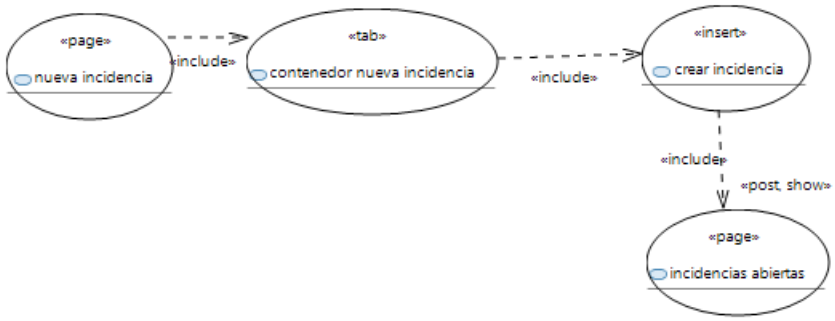
Distribución de casos de uso en componentes:

Asignamos la parte de interfície de usuario al componente frontoffice, la lógica de negocio al componente incidencias.



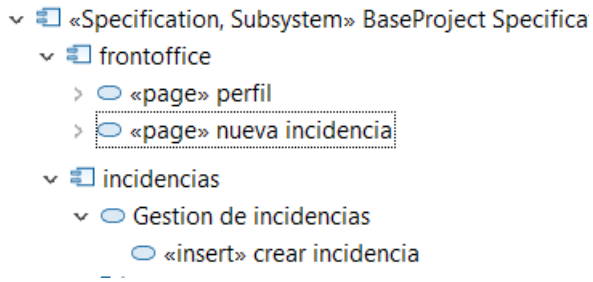
3.5.3 nueva incidencia

La página de nueva incidencia nos permitirá crear incidencias. Al crear la incidencia queremos ir a la página de incidencias abiertas.

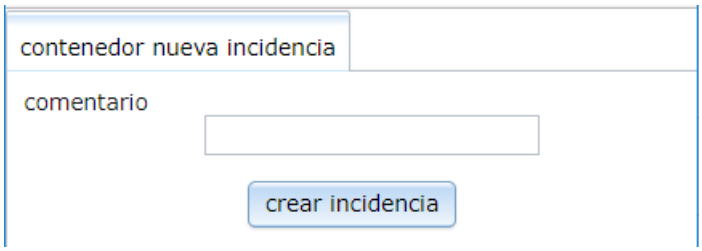


Distribución de casos de uso en componentes:

Separamos los casos de uso en componentes. La estructura de la interfície de usuario, en el componente frontoffice, y la lógica de negocio, en el componente incidencias, como caso de uso perteneciente al caso de uso de primer nivel (que representa un servicio) Gestión de incidencias.



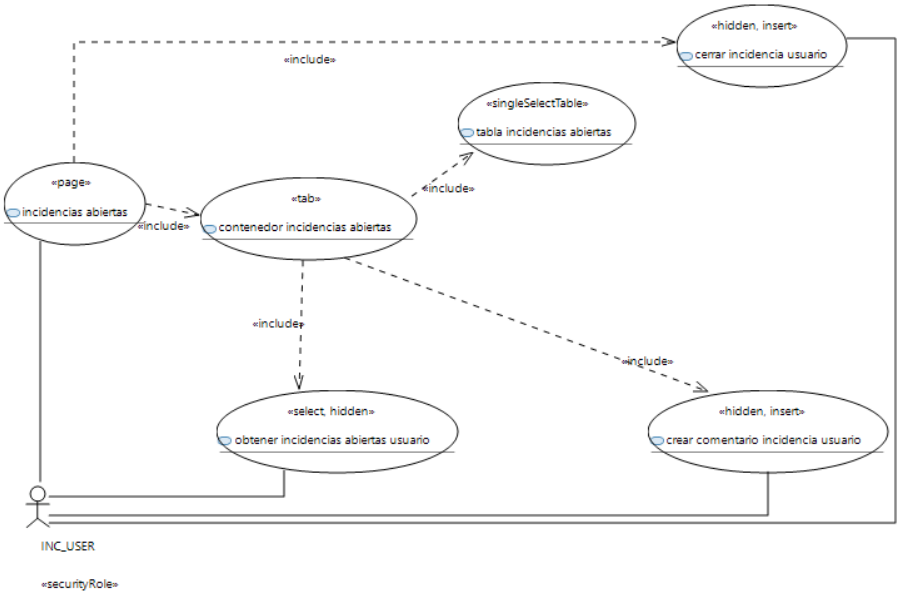
La siguiente imagen muestra la interfície resultante



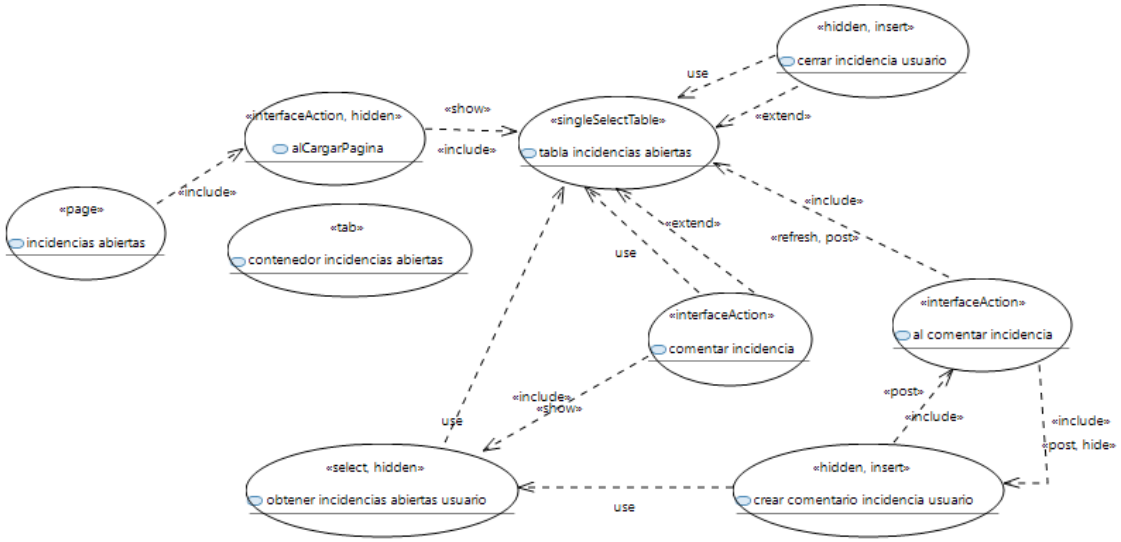
3.5.4 incidencias abiertas

La página de incidencias abiertas, mostrará un listado con las incidencias abiertas. Al seleccionar una de las incidencias, podrá elegir entre las opciones "comentar" y "cerrar". Al comentar, el estado de la incidencia pasará a estado "PENDING_COMPANY", al cerrar,la incidencia pasará a estado "CLOSED". Al modificar un registro, queremos que los datos de la tabla se actualicen, y los formularios utilizados queden ocultos.

Estructura estática:



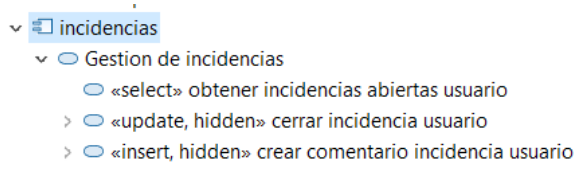
Comportamiento dinámico:

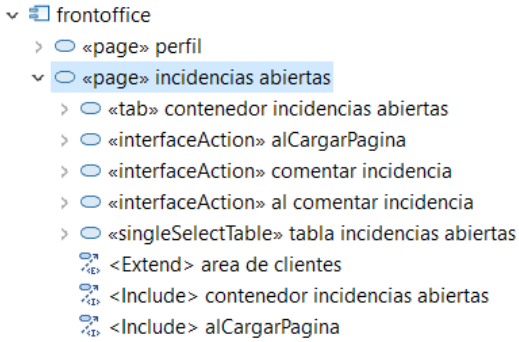


En el diagrama no se ve, pero la "tabla incidencias abiertas", tiene asignada como operación de obtención de datos el caso de uso "obtener incidencias abiertas usuario".

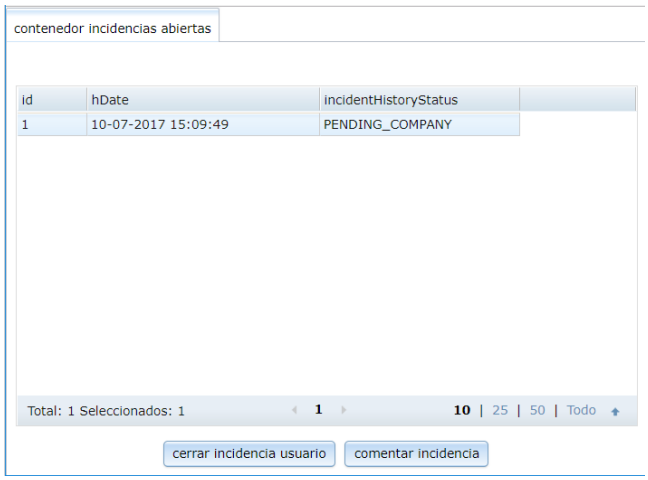
Distribución de casos de uso en componentes:

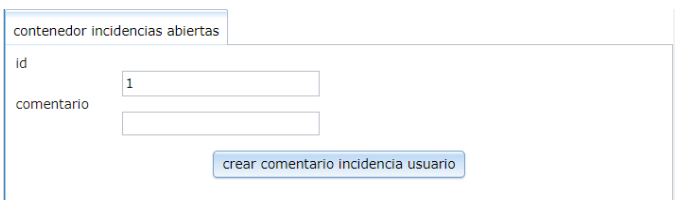
Asignamos la parte de interfície de usuario al componente frontoffice, la lógica de negocio al componente incidencias.





Las siguientes imágenes muestran la interfície resultante:





3.5.5 historial de incidencias

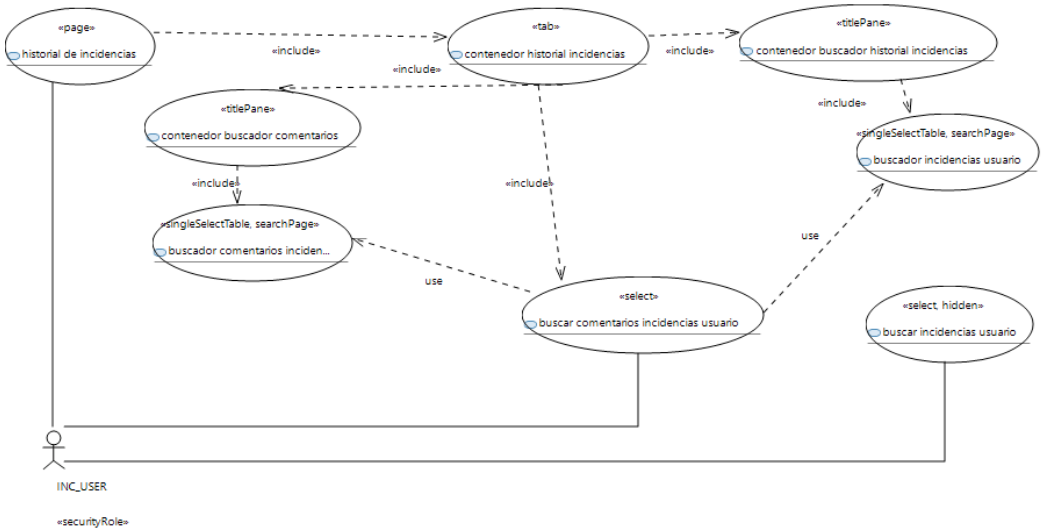
La página de historial de incidencias contendrá dos buscadores.

El buscador de historial de incidencias: Al seleccionar la incidencia, podrá elegir la opción "Ver historial incidencia", que le mostrará el historial de la incidencia.

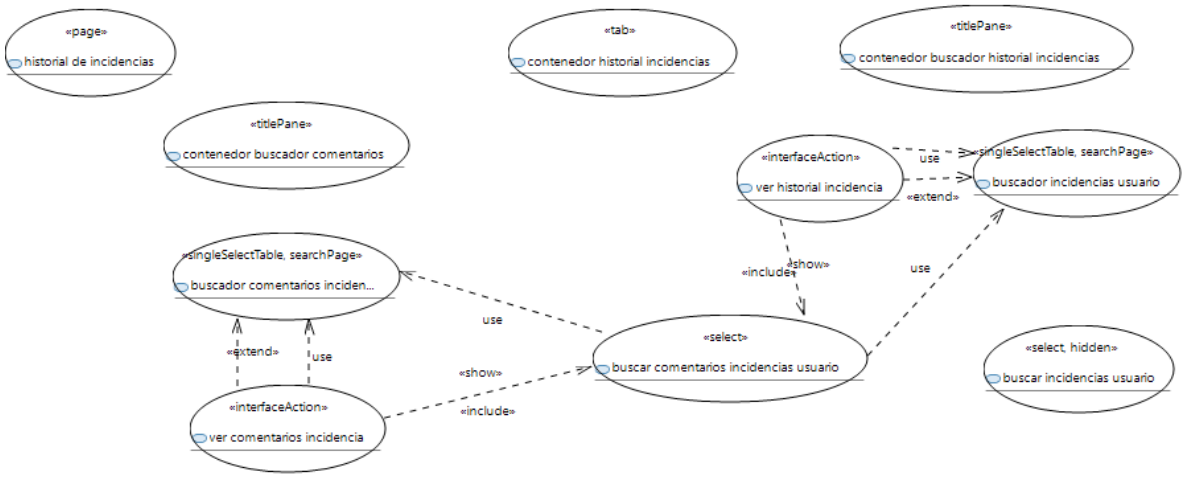
El buscador de comentarios en las incidencias: Al seleccionar un registro de la tabla, podrá escoger la opción "Ver comentarios".

Ambos formularios mostraran el historial de comentarios de la incidencia seleccionada tras la búsqueda.

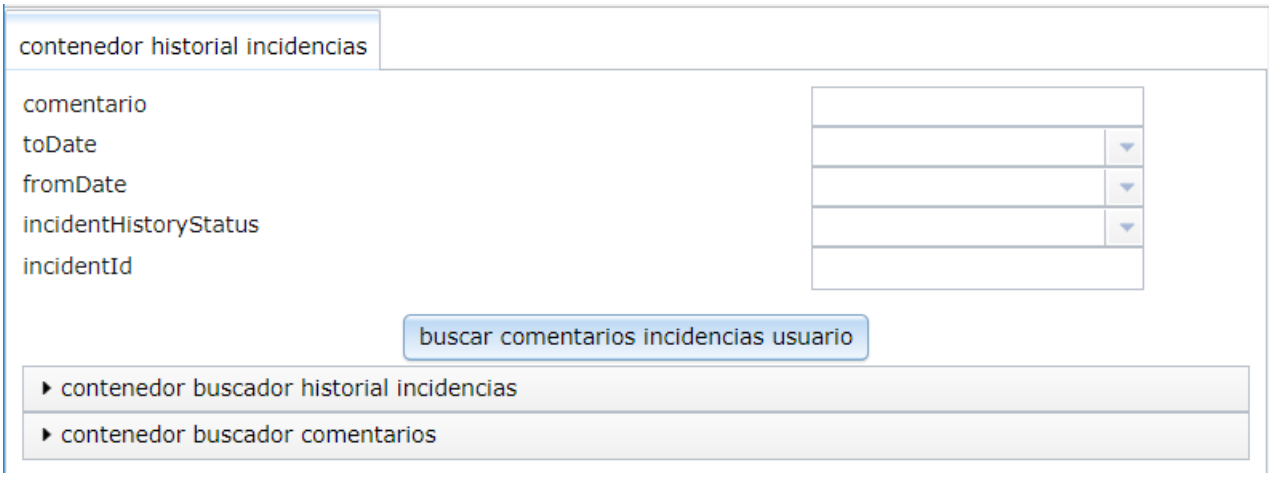
Estructura estática:



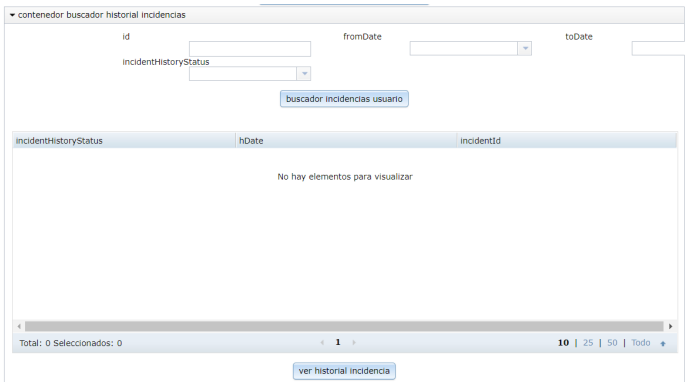
Comportamiento dinámico:

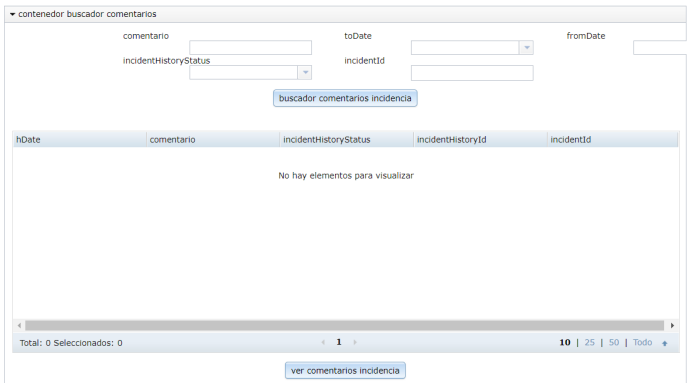


En el diagrama no se ve, pero la tabla "buscador incidencias usuario", tiene asignada como operación de obtención de datos el caso de uso "buscar incidencias usuario",y la tabla "buscador comentarios incidencias", el caso de uso "buscar comentarios incidencias usuario". Las siguientes imágenes muestran la interfície resultante:



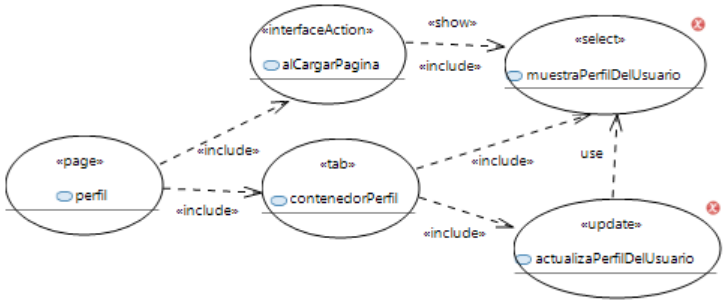
El formulario de búsqueda que se ve al principio, está generado por el select "buscar comentarios incidencias usuarios", que posteriormente podemos ocultar. Los contenidos de los title panes son los siguientes:





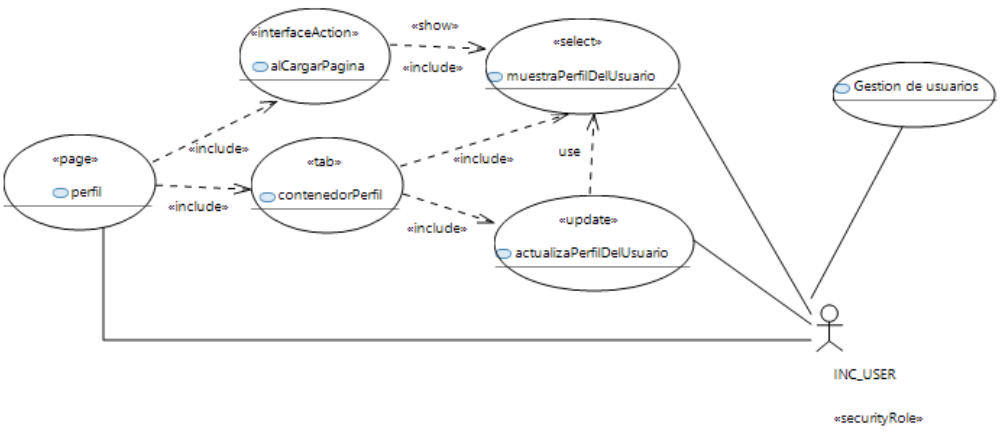
3.6 Validación de los casos de uso del sistema

Abrimos el documento *src/main/uml/proyecto_base.uml* con el "UML Model Editor", y siguiendo los pasos explicados en el módulo eclipse de Samsara Sfotware Comunity Edition validamos el modelo.



Vemos que hay errores en el modelo, y siguiendo las instrucciones dadas al colocar el ratón encima del icono de error, corregimos el modelo. Básicamente nos ha faltado asociar los casos de uso a los actores involucrados.

- Los casos de uso de primer nivel que representan servicios, tienen que estar asociados al mismo actor que todos los casos de uso que contenga.
- Los casos de uso que representan elementos de la interfaz de usuario, sólo tienen que estar asociados a actores si tienen el estereotipo <<interface::page>> aplicado.
- Los casos de uso que representan servicios utilizados en una página tienen que tener asociado al actor que tenga asociado la página, ya que sinó la página no tiene las mismas restricciones que sus funcionalidades.



3.7 Creación de los servicios

Ahora que ya tenemos creados los casos de uso, vamos a crear los servicios. Para ello, debemos ejecutar el comando *mvn clean install* en el generador ***m2m / cim / specification2implementation***

Si hay algun problema ejecutando la transformación, debemos revisar que las rutas utlizadas por los generadores son correctas, en el fichero *generators/pom.xml* Si las modificamos, debemos ejecutar el comando *mvn clean install* en el directorio generators, y luego volver a ejecutar la transformación.

Al finalizar la transformación, veremos que en el elemento *"pim" / <<Implement, Subsystem>> BaseProject Implementation / <<Service>> BaseProject* aparecen tantos componentes como componentes tiene el elemento *cim / <<Specification, Subsystem>> SpecificationComponent*, y que cada uno de los componentes contiene un paquete, con una interfaz con el nombre del caso de uso de primer nivel, y con tantas operaciones como casos de uso contiene el caso de uso que representa el servicio. También aparecen más paquetes "data.in", "data.out" y "exceptions", con clases con los nombres de las operaciones, que son los parámetros de entrada, salida y excepciones de cada operación.

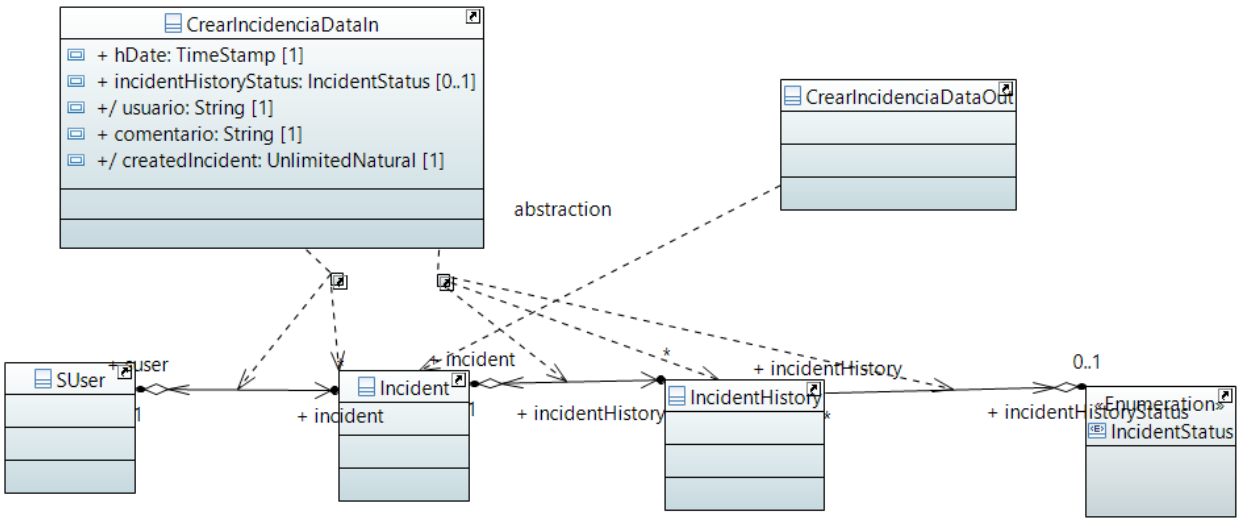
3.8 Mapeo de los formularios con la base de datos

Vamos a relacionar la información de la base de datos, con la información que se debe mostrar / actualizar / eliminar en una operación de un servicio. Para ello, creamos un diagrama para cada uno de los mapeos, por ejemplo creamos un diagrama con nombre "muestraPerfilDelUsuario DataMapping ClassDiagram" en la ruta *pim" / <<Implement, Subsystem>> BaseProject Implementation / <<Service>> BaseProject / <<Service>> usuarios / com.samsarasoftware.base_project.usuarios.services / gestiondeusuarios*

El primer paso, del mapeo sólo implica definir qué campos de entrada y salida hay que mostrar, su multiplicidad (obligatorio / no obligatorio) y de qué tablas se obtienen los datos, sin entrar en el detalle, por qué, porque tenemos una transformación que nos rellenará la mayoría de los valores del mapeo automáticamente. Vamos a crear primero las abstracciones y luego veremos los detalles.

3.8.1 nueva incidencia

Para crear una nueva incidencia, debemos hacer dos inserciones. Primero tenemos que crear una incidencia, y posteriormente crear un comentario asociado a esta nueva incidencia creada. Para hacer esto, debemos crear dos abstracciones desde la clase de entrada a las entities, tal y como muestra el siguiente gráfico:

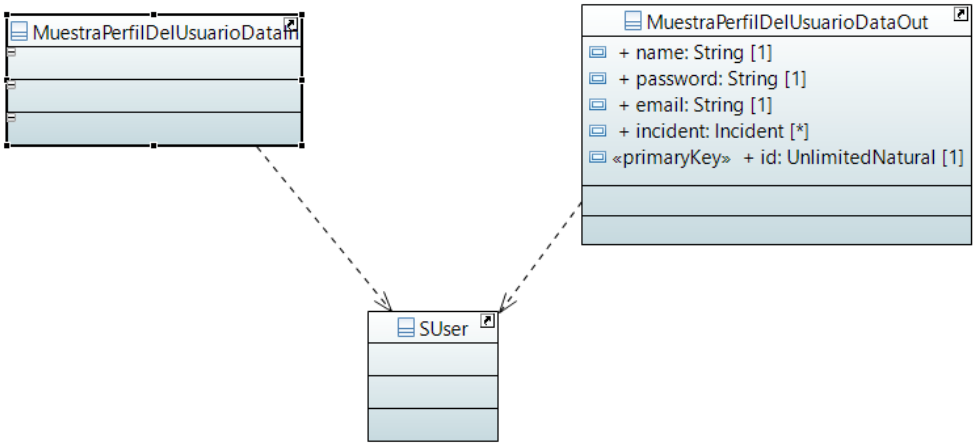


La abstracción de salida es necesaria, aunque la operación no devuelva resultados.

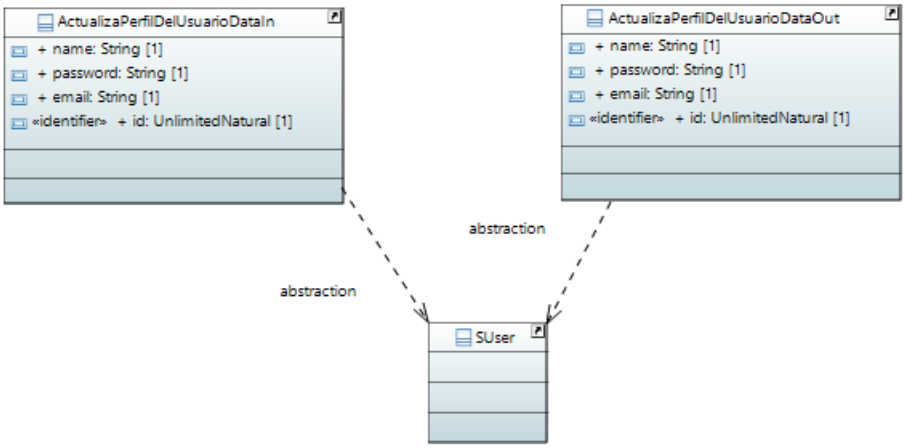
3.8.2 muestra perfil del usuario

La información de salida será toda la de la tabla *SUser*, así que hacemos un copy paste de todos los atributos de la clase *SUser* en la clase *MuestraPerfilDelUsuarioDataOut*.

Creamos una abstracción entre la clase de datos de entrada *MuestraPerfilDelUsuarioDataIn* y la tabla *SUser* (realmente no hay relación, pero es necesaria), y una abstracción entre la clase de datos de salida *MuestraPerfilDelUsuarioDataOut* y la tabla *SUser*.



3.8.3 actualiza perfil del usuario

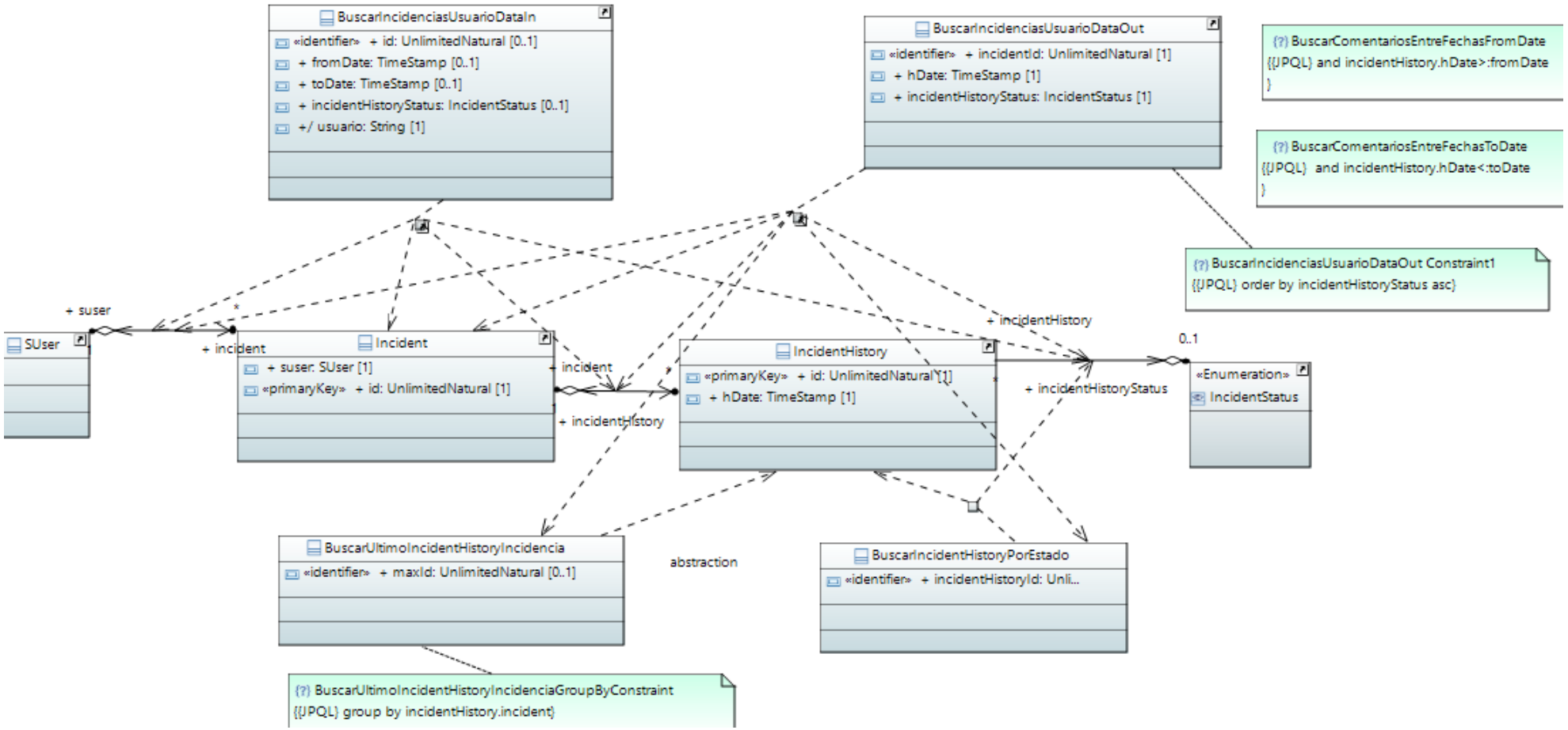


3.8.4 buscar incidencias usuarios

Este caso es bastante complejo, debemos mirar qué consulta debemos generar primero, para luego modelarla correctamente. Queremos obtener las incidencias cuyo último caso histórico tenga un estado concreto en unas fechas concretas.

```
SELECT
 incidentHistory.hDate
, incidentHistoryStatus.literal
, incident.id
FROM
com.samsarasoftware.base_project.incident.IncidentEntity incident
left join incident.suser  suser
left join incident.incidentHistory  incidentHistory
left join incidentHistory.incidentHistoryStatus  incidentHistoryStatus
WHERE
1=1
        //subquery BuscarUltimoIncidentHistoryIncidencia
        and incidentHistory.id in
        (       SELECT
                        max(incidentHistory.id)
                FROM
                        com.samsarasoftware.base_project.incident.IncidentHistoryEntity incidentHistory
                WHERE
                        1=1
                group by incidentHistory.incident
        )
        //subquery BuscarIncidentHistoryPorEstado
        and incidentHistory.id in
        (       SELECT
                        incidentHistory.id
                FROM
                        com.samsarasoftware.base_project.incident.IncidentHistoryEntity incidentHistory
                        left join incidentHistory.incidentHistoryStatus  incidentHistoryStatus
                WHERE
                        1=1
                        and incidentHistoryStatus.literal like :incidentHistoryStatus
        )
and incident.id like :id");
and suser.name like :usuario");
and incidentHistory.hDate>:fromDate
and incidentHistory.hDate<:toDate
order by incidentHistoryStatus asc
```

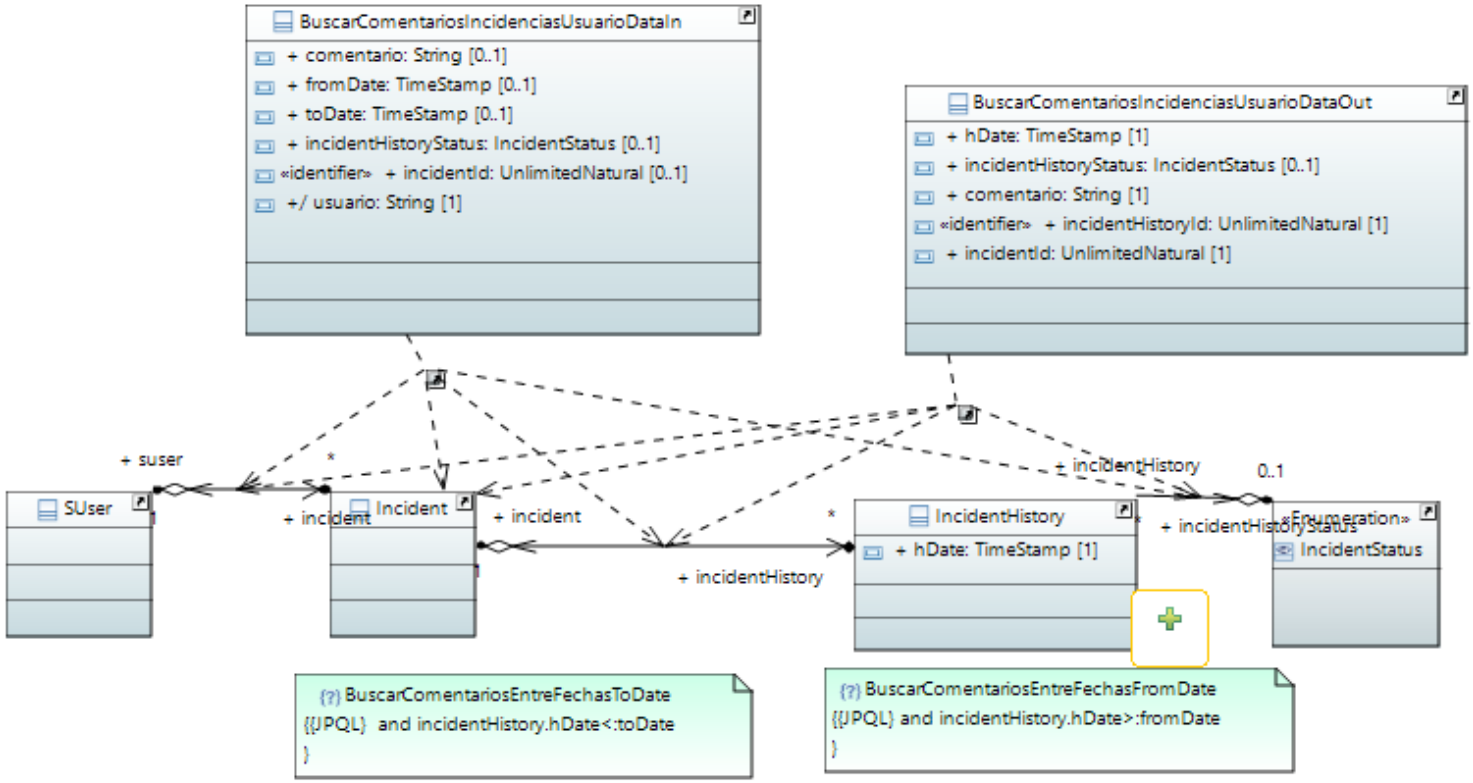
Tal como podemos ver, tenemos dos subqueries, y bastantes filtros, tanto en la query principal como en las subqueries.



Por ahora sólo establecemos los parámetros de entrada, de salida, y las abstracciones principales.

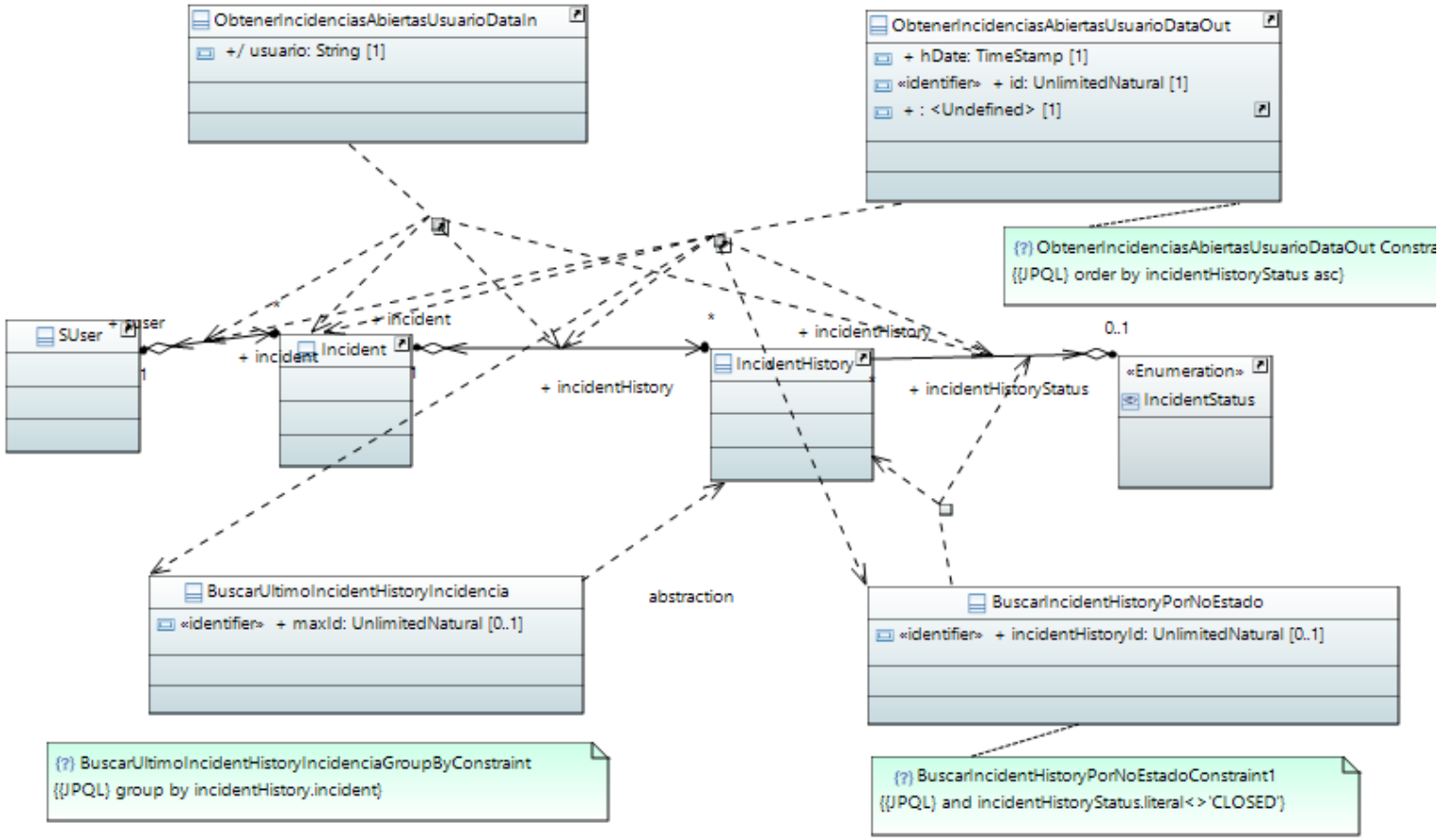
3.8.5 buscar comentarios incidencias usuarios

Este caso es más sencillo:

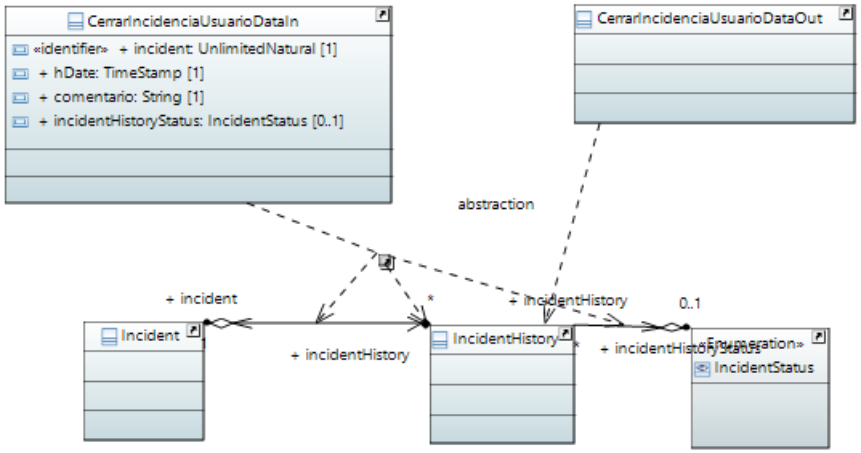


3.8.6 obtener incidencias abiertas usuario

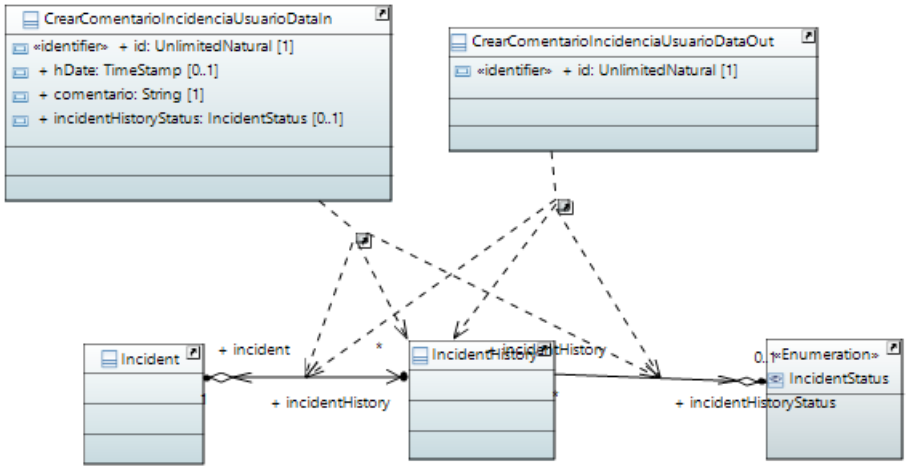
Otro caso con subqueries:



3.8.7 cerrar incidencia usuario



3.8.8 crear comentario incidencia



3.9 Asignación de valores

El siguiente paso es asignar un valor por defecto a todas las propiedades de entrada y salida de los mapeos, que es el que indica a qué campo de las tablas del mapeo corresponde.

3.9.1 Asignación automática

Disponemos de una transformación que nos puede agilizar bastante el trabajo de asignación de los valores de los mapeos. Para ello era necesario haber creado las abstracciones de entrada y salida, e indicar qué tablas estaban implicadas en la operación, y qué campos de entrada y salida vamos a utilizar.

Para ejecutar la transformación, debemos ejecutar el comando *mvn clean install* en el directorio *generators / pim / initializeDataMappingExpressions*.

3.9.2 nueva incidencia

Veremos que después de la transformación la mayoría de valores han sido asignados automáticamente. Debemos corregir.

De entrada:

- usuario: Es un valor derivado, que obtendremos del servicio de autenticación. Su mapeo es *suser.name*
- *hDate: Debe ser **ReadOnly**, ya que será la hora actual del servidor, y el cliente no puede establecer este valor. Su mapeo es incidentHistory.hDate*
- *incidentHistoryStatus: Debe ser **ReadOnly**, ya que será la hora actual del servidor, y el cliente no puede establecer este valor. Su mapeo es incidentHistoryStatus.literal*
- *createdIncident: Es un valor derivado, que estableceremos como el id de la incidencia, una vez creada. Su mapeo es incident.id*

3.9.3 muestra perfil del usuario

Veremos que después de la transformación la mayoría de valores han sido asignados automáticamente. Debemos corregir.

De entrada:

- usuario: Es un valor derivado, que obtendremos del servicio de autenticación. Su mapeo es *suser.name*

3.9.4 actualiza perfil del usuario

Los valores habrán sido asignados automáticamente.

3.9.5 buscar incidencias usuarios

Veremos que después de la transformación la mayoría de valores han sido asignados automáticamente. Debemos corregir.

De entrada:

- usuario: Es un valor derivado, que obtendremos del servicio de autenticación. Su mapeo es *suser.name*
- fromDate: no debe tener valor por defecto, ya que no hay una relación de mapeo directa, sólo se utilizará en las restricciones.
- toDate: no debe tener valor por defecto, ya que no hay una relación de mapeo directa, sólo se utilizará en las restricciones.
- incidentHistoryStatus: Se va a utilizar en la subquery BuscarIncidentHistoryPorEstado. Su valor por defecto es *BuscarIncidentHistoryPorEstado.incidentHistoryStatus.literal*

3.9.6 buscar comentarios incidencias usuarios

Veremos que después de la transformación la mayoría de valores han sido asignados automáticamente. Debemos corregir.

De entrada:

- usuario: Es un valor derivado, que obtendremos del servicio de autenticación. Su mapeo es *suser.name*
- incidentId: Su mapeo es incident.id
- fromDate: no debe tener valor por defecto, ya que no hay una relación de mapeo directa, sólo se utilizará en las restricciones.
- toDate: no debe tener valor por defecto, ya que no hay una relación de mapeo directa, sólo se utilizará en las restricciones.

De salida:

- incidentId: Su mapeo es incident.id

3.9.7 obtener incidencias abiertas usuario

Veremos que después de la transformación la mayoría de valores han sido asignados automáticamente. Debemos corregir.

De entrada:

- usuario: Es un valor derivado, que obtendremos del servicio de autenticación. Su mapeo es *suser.name*

3.9.8 cerrar incidencia usuario

Los valores habrán sido asignados automáticamente. Debemos corregir.

De entrada:

- incident: Su mapeo es *incident.id*
- hDate: Debe ser **ReadOnly**, ya que será la hora actual del servidor, y el cliente no puede establecer este valor.

3.9.9 crear comentario incidencia

Los valores habrán sido asignados automáticamente. Debemos corregir.

De entrada:

- comentario: Debe ser **ReadOnly**, ya que será una cadena predefinida, y el cliente no puede establecer este valor.
- hDate: Debe ser **ReadOnly**, ya que será la hora actual del servidor, y el cliente no puede establecer este valor.
- incidentHistoryStatus: Debe ser **ReadOnly**, ya que definimos a qué estado pasará la incidencia, y el cliente no puede establecer este valor.

3.10 Creación de las subqueries

3.10.1 buscar incidencias usuarios

Debemos crear las subqueries:

- BuscarUltimoIncidentHistoryIncidencia:
  - maxId: su mapeo es *max(incidentHistory.id)*

- BuscarIncidentHistoryPorEstado
  - incidentHistoryId: su mapeo es *incidentHistory.id*

Ahora, debemos asignar el siguiente mapping a la abstracción de salida, como un OpaqueExpression en lenguaje JPQL:

and incidentHistory.id in :BuscarUltimoIncidentHistoryIncidencia and incidentHistory.id in :BuscarIncidentHistoryPorEstado

3.10.2 obtener incidencias abiertas usuario

Debemos crear las subqueries:

- BuscarUltimoIncidentHistoryIncidencia:
  - maxId: su mapeo es *max(incidentHistory.id)*
- BuscarIncidentHistoryPorNoEstado
  - incidentHistoryId: su mapeo es *incidentHistory.id*

Ahora, debemos asignar el siguiente mapping a la abstracción de salida, como un OpaqueExpression en lenguaje JPQL:

and incidentHistory.id in :BuscarUltimoIncidentHistoryIncidencia and incidentHistory.id in :BuscarIncidentHistoryPorNoEstado

3.11 Asignación de restricciones

Ahora el paso final, es asignar las restricciones:

3.11.1 muestra perfil del usuario

No tiene restricciones.

3.11.2 actualiza perfil del usuario

No tiene restricciones.

3.11.3 buscar incidencias usuarios

Debemos crear una Constraint, sin contexto por cada restricción, y asignarle como contrainedElement el elemento que se indica.

- Creamos una constraint. Le asignamos como constrained element la clase *BuscarUltimoIncidentHistoryIncidencia.* La especificación en lenguaje JPQL es: *group by incidentHistory.incident*.
- Creamos una constraint. Le asignamos como constrained element la clase *BuscarIncidenciasUsuarioDataOut.* La especificación en lenguaje JPQL es: *order by incidentHistoryStatus asc*.
- Creamos una constraint. Le asignamos como constrained element el atributo *fromDate* de la clase *BuscarIncidenciasUsuarioDataIn.* La especificación en lenguaje JPQL es: *and incidentHistory.hDate>:fromDate*.
- Creamos una constraint. Le asignamos como constrained element el atributo *toDate* de la clase *BuscarIncidenciasUsuarioDataIn.* La especificación en lenguaje JPQL es: *and incidentHistory.hDate<:toDate*.

3.11.4 obtener incidencias abiertas usuario

Debemos crear una Constraint, sin contexto por cada restricción, y asignarle como contrainedElement el elemento que se indica.

- Creamos una constraint. Le asignamos como constrained element la clase *BuscarUltimoIncidentHistoryIncidencia.* La especificación en lenguaje JPQL es: *group by incidentHistory.incident*.
- Creamos una constraint. Le asignamos como constrained element la clase *BuscarIncidentHistoryPorNoEstado.* La especificación en lenguaje JPQL es: *and incidentHistoryStatus.literal<>'CLOSED'*
- Creamos una constraint. Le asignamos como constrained element la clase *ObtenerIncidenciasAbiertasUsuarioDataOut.* La especificación en lenguaje JPQL es: *order by incidentHistoryStatus asc*.

3.11.5 buscar comentarios incidencias usuarios

Debemos crear una Constraint, sin contexto por cada restricción, y asignarle como contrainedElement el elemento que se indica.

- Creamos una constraint. Le asignamos como constrained element el atributo *fromDate* de la clase *BuscarComentariosIncidenciasUsuarioDataIn.* La especificación en lenguaje JPQL es: *and incidentHistory.hDate>:fromDate*.

- Creamos una constraint. Le asignamos como constrained element el atributo *toDate* de la clase *BuscarComentariosIncidenciasUsuarioDataIn.* La especificación en lenguaje JPQL es: *and incidentHistory.hDate<:toDate*.

4 Especificando la arquitectura del sistema de soporte

Ahora que tenemos la arquitectura de componentes del sistema, debemos definir el PSM o modelo de la arquitectura del sistema de soporte, es decir, los componentes reales a generar.

Para ello utilizaremos la transformación psm que más nos convenga. En nuestro caso, vamos a utilizar la arquitectura J2EE con las siguientes tecnologías y restricciones:

- Lógica de negocio en EJB3
- Transacciones JTA, y seguridad controladas por el contenedor EJB3
- Base de datos Oracle
- Interfaz de usuario utilizando HTML5 y Dojotoolkit
- Protocolo de comunicación entre la capa de presentación y la capa de negocio JSON-RPC2 sobre HTTP utilizando servlets.

Para ello disponemos de la transformación *generators / m2m / psm / initializePsmXaDisk*. Ejecutamos el comando *mvn clean install* en este directorio, y veamos los resultados.

5 Generando el código

Como es la primera vez que generamos el código, debemos ejecutar el comando *mvn clean install* en el directorio *generators / mof2text / invokers*.

Veremos que aparece un nuevo directorio base_project con los componentes maven ya generados. Ejecutamos el comando *mvn clean install* en este directorio para compilar el código de la aplicación, y veamos dónde falla.

5.1 Propiedades read-only

Las propiedades read-only no tienen un valor asignado, debemos asignar los siguientes valores:

- com.samsarasoftware.base_project.incidencias.services.gestiondeincidencias.data.in.CrearIncidenciaDataIn
  - getIncidentHistoryStatus: returns "CREATED";
- com.samsarasoftware.base_project.incidencias.services.gestiondeincidencias.data.in.CerrarIncidenciaUsuarioDataIn
  - getComentario: returns "Cierre de incidencia";
  - getHDate: return new java.util.date();
  - getIncidentHistoryStatus: returns "CLOSED";
- com.samsarasoftware.base_project.incidencias.services.gestiondeincidencias.data.in.CrearComentarioIncidenciaUsuarioDataIn:
  - getHDate: returns new java.util.Date();
  - getIncidentHistoryStatus: returns "PENDING_COMPANY";

5.2 Propiedades derived

Las propiedades derived no tienen valor, y debe asignarse entre las porciones de código de usuario, que se mantendrá cuando se modifique el modelo uml, o se vuelva a generar el código.

```
//Start of user code ...-query-params
//End of user code
```

- com.samsarasoftware.base_project.incidencias.services.gestiondeincidencias.ejb.GestionDeIncidenciasServiceImpl:
  - crearIncidencia: Tenemos dos bloques de código personalizado
    - *//Start of user code derivedValues for input abstraction crearIncidencia 1*

      crearIncidenciaDataIn.setUsuario(context.getCallerPrincipal().getName());

      //End of user code
    - *//Start of user code derivedValues for input abstraction crearIncidencia 2*

      crearIncidenciaDataIn.setCreatedIncident(incident.id);

      //End of user code
  - buscarComentariosIncidenciasUsuario:

//Start of user code buscarComentariosIncidenciasUsuario-query-params

buscarComentariosIncidenciasUsuarioDataIn.setUsuario(context.getCallerPrincipal().getName());

//End of user code

- buscarIncidenciasUsuario:

  //Start of user code buscarIncidenciasUsuario-query-params

  buscarIncidenciasUsuarioDataIn.setUsuario(context.getCallerPrincipal().getName());

  //End of user code

- obtenerIncidenciasAbiertasUsuario:

  //Start of user code obtenerIncidenciasAbiertasUsuario-query-params

  obtenerIncidenciasAbiertasUsuarioDataIn.setUsuario(context.getCallerPrincipal().getName());

  //End of user code

5.3 Compilar las modificaciones

Ahora es necesario volver a ejecutar el comando *mvn clean install* en el directorio base_project_test\base_project hasta que veamos que no hay errores de compilación.

6 Creando la base de datos

Para crear la base de datos, debemos conectarnos a la base de datos, y ejecutar los siguientes scripts:

- workspace\base_project_test\base_project\base_project-jpa\src\main\scripts\oracle\create_tables_com.samsarasoftware.base_project.sql
- workspace\base_project_test\base_project\base_project-jpa\src\main\scripts\oracle\create_foreignKeys_com.samsarasoftware.base_project.sql
- workspace\base_project_test\base_project\base_project-jpa\src\main\scripts\oracle\create_sequences_com.samsarasoftware.base_project.sql

- workspace\base_project_test\base_project\base_project-jpa\src\main\scripts\oracle\create_indexes_com.samsarasoftware.base_project.sql
- workspace\base_project_test\base_project\base_project-jpa\src\main\scripts\oracle\initialize_tables_com.samsarasoftware.base_project.sql
- Hacer commit.

7 Desplegar la aplicación en el servidor de aplicaciones

7.1 Configurar el datasource

Debemos crear un fichero de definición de datasource para el servidor de aplicaciones, en la ruta *jboss-5.1.0.GA\server\jboss01\deploy*. Podemos aprobechar el datasorce generado en *workspace\base_project_test\base_project\base_project-jpa\src\main\scripts\datasources*, o definir nuestro datasorce de nuevo:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
	<local-tx-datasource>
		<jndi-name>com.samsarasoftware.base_project.db</jndi-name>
		<!--Start of user code datasource-connection-params-->
		<connection-url>jdbc:oracle:thin:@127.0.0.1:1521:xe</connection-url>
		<driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
		<user-name>DB_USER</user-name>
		<password>DB_USER</password>
		<!-- End of user code -->
		<new-connection-sql> BEGIN EXECUTE IMMEDIATE 'ALTER SESSION
SET	CURRENT_SCHEMA=DB_USER'; END; </new-connection-sql>
		<min-pool-size>1</min-pool-size>
		<max-pool-size>20</max-pool-size>
	</local-tx-datasource>
</datasources>
```

7.2 Configurar propiedades de sistema

Debemos establecer propiedades del sistema, utilizadas por la aplicación.

- com.samsarasoftware.base_project.repository.path : Especifica la ruta donde se creará el repositorio xa-disk utilizado en la persistencia de ficheros.

Para ello, modificaremos el fichero *jboss-5.1.0.GA\server\jboss01\conf\jboss-service.xml*

Añadimos el siguiente código al final del fichero:

```
<mbean code="org.jboss.varia.property.SystemPropertiesService"
name="jboss:type=Service,name=TutorialSystemProperties">
        <attribute name="Properties">
com.samsarasoftware.base_project.repository.path=/C:/app/samsarasoftware/base_project
        </attribute>
</mbean>
</server>
```

7.3 Despliegue del ear

Para que las rutas JNDI generadas sean las esperadas, el ear que despleguemos tiene que tener por nombre base_project.ear, es decir, será el nombre del ear generado, sin la cadena de la versión.

Debemos copiar el fichero *workspace\base_project_test\base_project\base_project-ear\target\base_project-0.0.1-SNAPSHOT.ear* en el directorio *jboss-5.1.0.GA\server\jboss01\deploy*

8 Validación de la aplicación

Para validar el funcionamiento de la aplicación, primero debemos crear el usuario de la aplicación con el que queremos autenticarnos. Para ello, debemos ejecutar el siguiente comando en el usuario DB_USER de la base de datos:

```
Insert into SUSER (EMAIL,ID,NAME,PASSWORD) values (
'info@samsara-software.es','1','pere','pere'
);
```

Después, ya podemos validar la aplicación en la URL http://localhost:28080/BaseProject/home/public/index/index.html y autenticarnos con el usuario "pere", contraseña "pere".

9 Modificación de la interfície de usuario

La interfície de usuario generada es un prototipo rápido de la aplicación final. Todavía queda reposicionar elementos , cambiar estilos, etc...

Una de las interfícies generadas, que no es exactamente como queremos es la del historial de incidencias. Vamos a modificarla, para que quede a nuestro gusto. Para ello, vamos a seguir los pasos "proceso de instalación" y "Arranque de maqetta y creación del usuario" explicados en el módulo de maqetta del producto Samsara Software Community Edition.

Una vez instalada la aplicación maqetta, y creado un usuario, configuraremos el entorno de trabajo para la aplicación que estamos generando.

- Debe crearse un directorio workspace como directorio raíz del proyecto, que representará el contexto raíz de la web (el dominio en la URL de plublicación).
- Dentro del directorio workspace debemos crear los directorios necesarios para formar el contexto (o URL) de cada componente web, tal y como se ha configurado en el taggedValue "http::http::contextPath" de los componentes estereotipados con "http::http" en el PSM:
  - BaseProject
  - BaseProject/frontoffice
- Configuración del fichero .settings/libs.settings:Configuración del fichero .settings/libs.settings:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<libraries>
<library id="dojo" name="dojo" version="1.8" virtualRoot="workspace/BaseProject/lib/dojo"/>
<library id="DojoThemes" name="DojoThemes" required="true" version="1.8"
virtualRoot="workspace/BaseProject/lib/dojo/themes"/>
<library id="gridx" name="gridx" version="1.0.0" virtualRoot="workspace/BaseProject/lib/dojo/gridx"/>
<library id="html" name="html" required="true" version="0.8"/>
<library id="clipart" name="clipart" version="1.0" virtualRoot="workspace/BaseProject/lib/clipart"/>
<library id="maqettaSamples" name="maqettaSamples" required="true" version="1.0" virtualRoot="samples"/>
<library id="maqetta" name="maqetta" required="true" version="0.3" virtualRoot="workspace/BaseProject/lib/maqetta"/>
<library id="shapes" name="shapes" version="1.0" virtualRoot="workspace/BaseProject/lib/shapes"/>
<library id="zazl" name="zazl" required="true" version="0.3.0" virtualRoot="workspace/BaseProject/lib/zazl"/>
<library id="samsarasoftware" name="samsarasoftware" version="0.0.2"
virtualRoot="workspace/BaseProject/lib/samsarasoftware"/>

</libraries>
```

- Configuración del fichero .settings/davinci.ui.ProjectPrefs.settings:

```
{"themeFolder":"./workspace/BaseProject/lib/dojo/themes","widgetFolder":"./lib/custom"}
```

Ahora, tenemos configurado el entorno de trabajo para el componente BaseProject, pero nos hace falta añadir los documentos html de cada componente de interfaz de usuario:

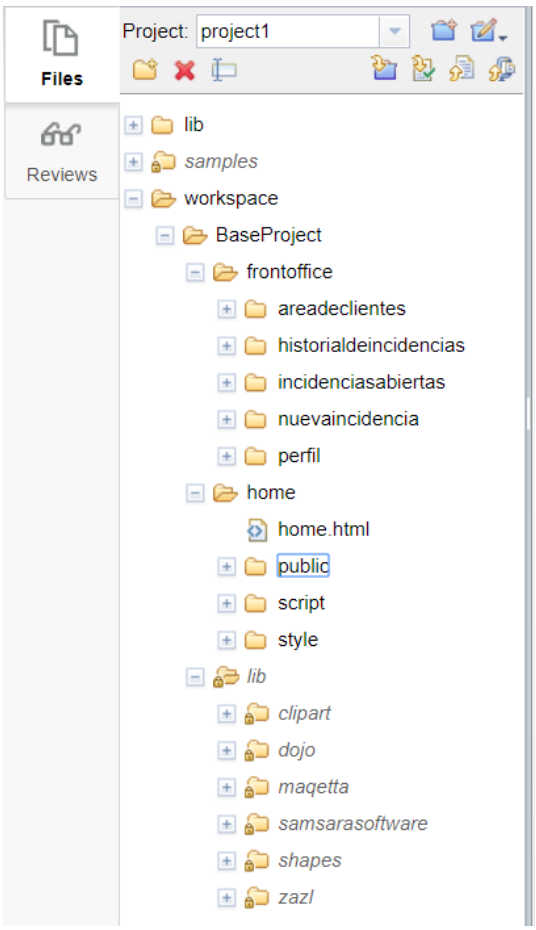
Para el componente BaseProject:

- Copiamos el contenido de la carpeta del código generado *base_project/BaseProject-dojo/src/main/webapp* en el directorio *%MAQETTA_DIR %/users/a/workspace/BaseProject*

Para el componente frontoffice.

- creamos el directorio *%MAQETTA_DIR %/users/a/workspace/BaseProject/frontoffice*
- Copiamos el contenido de la carpeta del código generado *base_project/frontoffice-dojo/src/main/webapp* en el directorio que acabamos de crear.

Para que maqetta pueda aplicar los cambios, es necesario cerrar la sesión y volver a auntenticarse. La estructura de directorios que veremos en maqetta es la siguiente:



9.1 workarounds para bugs de maqetta

Por desgracia, maqetta tiene algunos bugs que quién sabe si algún día serán resueltos. De momento hemos detectado los siguientes, y explicamos como evitarlos a continuación

9.1.1 Los atributos no pueden estar contenidos en más de una línea

Para que el código sea más legible, por ejemplo, en la configuración de un elemento gridx, el atributo de la configuración se genera en más de una línea, pero maqetta no lo soporta bién, y realiza cambios que cambian la estructura y comportamiento de la página. Es por ello necesario aplicar el siguiente workaround a los ficheros que vayamos a editar con maqetta:

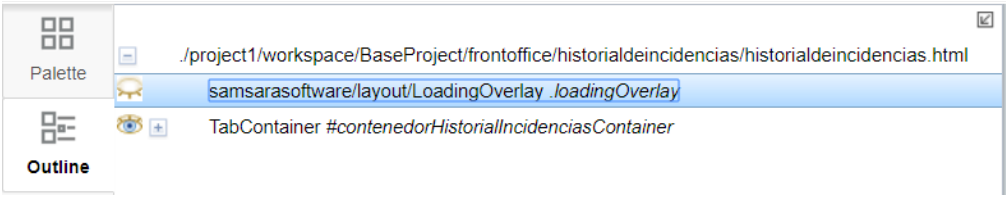
- Abrir el documento html a modificar en eclipse
- Hacer el siguiente replace, marcando el checkbox de "expresión regular":

- buscar: "**(^[^<^\n]+)\r\n**" (sin las comillas)
- reemplazar por: "" (sin las comillas)

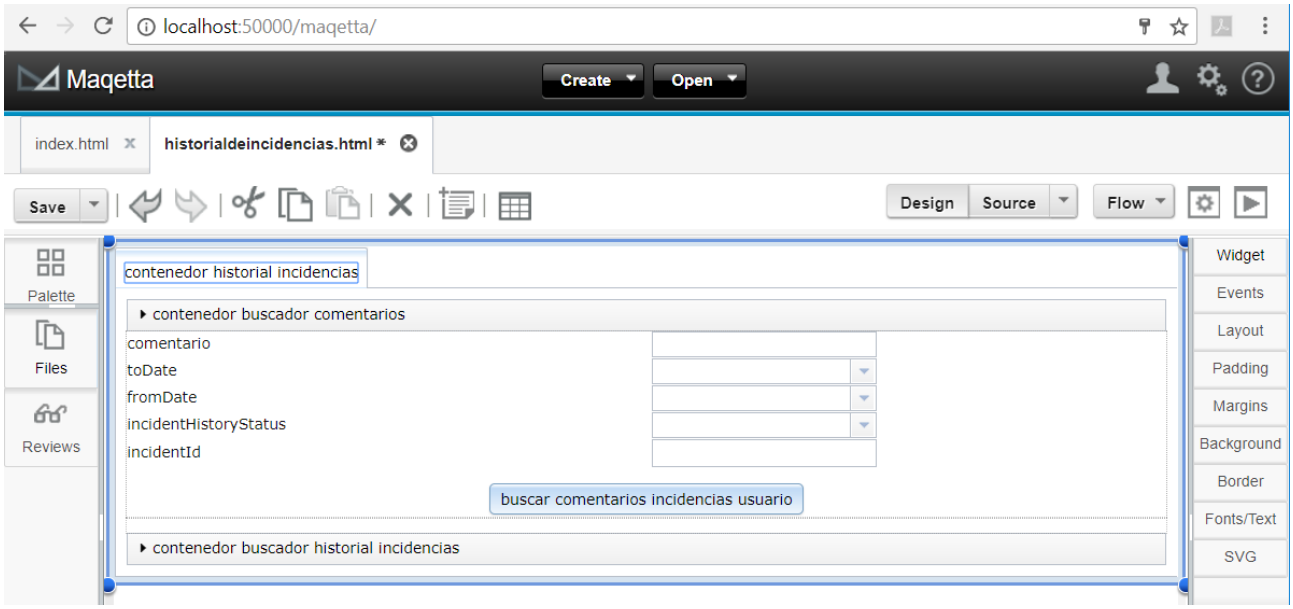
- repetir este paso hasta que no se encuentren ocurrencias.

9.2 historial de incidencias

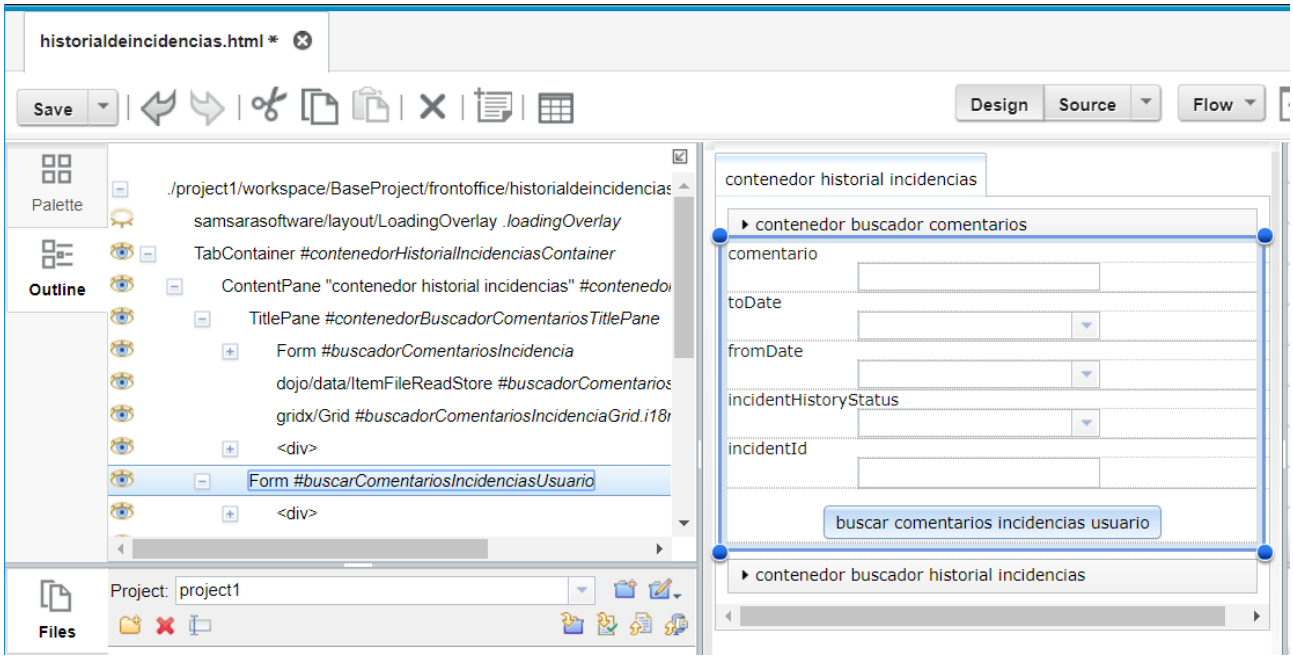
Al abrir la página, no veremos nada. Esto es debido a que la página tiene una capa que oculta el contenido hasta que no se haya cargado todo el contenido de ésta. Para poder editar la página, debemos ir al panel "outline", y ocultar la capa, tal y como se muestra en la siguiente imagen.



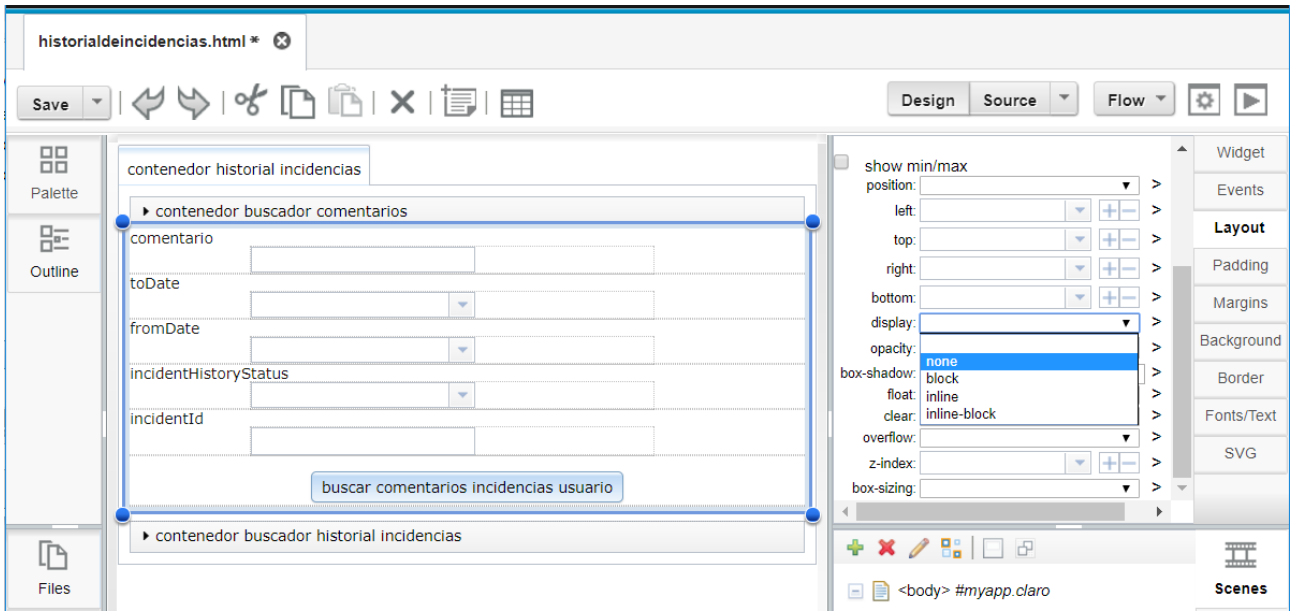
Ahora ya veremos el contenido de la página generada:



Lo que queremos hacer es ocultar el formulario de consulta que se genera cuando hay un caso de uso <<transaction::select>> sin otro caso de uso (insert, update, insertOrUpdate, delete) que lo use.



Para ello, modificamos el valor de display a "none" en la pestaña de propiedades "layout":



Ahora que ya hemos realizado el cambio, vamos al código fuente del documento html, y compiamos todo el contenido del tag <body> que sigue al LoadingOverlay en el documento html original. Si copiásemos todo el fichero, se producirían algunos desajustes, ya que maqetta realiza sus propias modificaciones para editar la página, es por eso que sólo hay que copiar el contenido del tag <body>. Tampoco copiamos el LoadingOverlay, porque le hemos cambiado la visibilidad. Podemos copiarlo si nos acordamos de dejarla visible, tal y como estaba originalmente.