**Arquitectura de Software**

**Experimento 3**



**Realizado por:**

|  |  |  |
| --- | --- | --- |
| **Persona** | **Rol** | **CódigoUniandes** |
| Carlos Ernesto González Vargas | Ingeniero de Requerimientos | 200819123 |
| Sandra Milena Gómez Ríos | Ingeniero de Producción | 201110951 |
| Andrés Mauricio Erazo Benavides | Ingeniero de Soporte | 201110949 |
| David Pérez Chibuque | Ingeniero de Calidad | 201117818 |
| Willian Alejandro Idrobo Luna | Arquitecto Jefe | 201110544 |
| Erik Fernando Arcos Franco | Ingeniero de Administración | 201110856 |

 

**Contenido**

**Pág.**

[**1.** **Definición del Sistema** 3](#_Toc298093087)

[**1.1.** **Vista Funcional** 4](#_Toc298093088)

[**1.2.** **Vista Despliegue** 5](#_Toc298093089)

[**1.3.** **Identificación de patrones** 6](#_Toc298093090)

[**2.** **Experimento A – Comunicación con Web Service** 7](#_Toc298093091)

[**3.** **Experimento B – Comunicación Asíncrona – Envío de correo electrónico** 12](#_Toc298093092)

[**4.** **Experimento C – Video llamada** 14](#_Toc298093093)

[**5.** **Referencias** 17](#_Toc298093094)

**Arquitectura de Software**

**Experimento 3**

1. **Definición del Sistema**

La empresa **Alpes Smart Home (ASH)** desea entrar a participar en el mercado de las casas inteligentes dado el considerable avance de la tecnología en dispositivos móviles, sensores inalámbricos y sistemas embebidos que ha habido durante los últimos años. Por esta razón la empresa **ASH** plantea el proyecto ***Casa Inteligente***, el cual surge para dar respuesta al diseño de arquitectura de solución de un sistema que permita la administración de viviendas inteligentes.

El sistema se debe cumplir con las necesidades iniciales del sistema, descritas a continuación:

* Analizar la información enviada por los sensores de humo, ventanas y puertas de las oficinas y viviendas monitoreadas.
* Una vez se detecta humo en una posición específica de la vivienda u oficina, se procede a notificar a los servicios de control de emergencias. Si se detecta que la vivienda está habitada, se informa a los bomberos y servicios de ambulancias, el número estimado de personas que están en ese momento en el lugar, así como las posibles ubicaciones.
* Si se trata de un sensor de ventana o puerta abierta sin autorización, se debe notificar a la policía sobre el evento.
* La comunicación con las entidades externas se realizará por medio de web services suministrados por dichas entidades.
* Generar un reporte consolidado mostrando todos los eventos registrados (ordenadas por tipo de evento) de forma cronológica (hora, día, mes, año).
* Evitar consulta y modificación de información no autorizadas. También deberá evitar suplantación de servicio y ante el fallo de un componente de sistema se debe continuar respondiendo solicitudes sin perder el estado del sistema.
* Por cada inmueble u oficina se tienen en promedio 50 tags RFID enviando información cada segundo a cuatro antenas ubicadas en diferentes posiciones de la casa u oficina. Estas antenas a su vez consolidan la información y la envían a la central.
* Cuando se detecta que un tag RFID ha violado una de las reglas definidas para su comportamiento, se debe generar una alarma indicando a todas las porterías y a los responsables de la vivienda u oficina lo ocurrido.

Para el experimento descrito a continuación se tiene el siguiente alcance:

* Definir un esquema xml para el intercambio de información con entidades externas.
* Crear un web service en una tecnología distinta a la del sistema central, para que éste sea consumido desde el sistema central.
* Definir una excepción en caso de que la notificación (comunicación con el web service) no pueda realizarse.
* Permitir la notificación de eventos relevantes por medio de correo electrónico.
* En caso de que ocurra un evento en la casa, se debe empezar a trasmitir video usando el API de aplicaciones como Skype.
  1. **Vista Funcional**



Figura 1. Vista Funcional

* 1. **Vista Despliegue**



Figura 2. Vista Despliegue

* 1. **Identificación de patrones**

Para el desarrollo de este sistema se tuvieron en cuenta los siguientes patrones:

***Capas***

Se aplica en el sistema central para separar las responsabilidades del sistema y favorecer la flexibilidad, interoperabilidad.

***Acceso a recursos – temprana***

Se aplica en la parte de obtención de la conexión a la base de datos para la generación de reportes.

***Acceso a recursos – caché***

Es aplicada para el manejo de y la información de los usuarios y las reglas que son configuradas para cada casa u oficina.

***Manejo de eventos – respuesta reactiva a eventos***

Al llegar un evento al sistema se delega la responsabilidad de manejar el evento a un hilo que se encargue de procesarlo de acuerdo a las reglas configuradas.

***Concurrencia y sincronización – coordinador***

Existe un componente receptor de todos los eventos que se encarga de crear una instancia de un manejador que procesa el evento de acuerdo a su tipo.

**Autenticación / Autorización**

La generación del reporte requiere que el usuario cuente con el rol de administrador. Es por esto que se usa tácticas de autenticación / autorización.

Se implementó parte de la estructura de PACE para manejar la seguridad del sistema.

**Mantener datos confidenciales**

El password de la cuenta del usuario se reconoce como información sensible y debe ser manejado como tal. Es por esto que se usa tácticas de cifrado de datos (hash) para conservar la confidencialidad de este campo.

**Firmas Digitales**

Para evitar la suplantación de identidad y verificar la legitimidad de fuente de información se implementó la táctica de firmas digitales.

**Redundancia Activa**

Para favorecer alta disponibilidad se hace uso de balanceo de cargas a servidores distintos usando el algoritmo round robin.

1. **Experimento A – Comunicación con Web Service**

El objetivo de este experimento es comunicar sistemas que funcionen bajo distintas plataformas. De manera más específica, desde el sistema central (Bajo plataforma java) se consumirá un web servicie publicado por un sistema externo (Bajo plataforma .Net). Dicha comunicación se hará a través del esquema XML definido.

Tabla 1. Descripción del Experimento A

|  |  |
| --- | --- |
| **Descripción del Experimento** | |
| ***Título*:** Interoperabilidad – Comunicación a través de web service entre sistemas. | ***ID*:** A1 |
| ***Descripción*:** Este experimento busca realizar una comunicación entre sistemas desarrollados bajo distintas plataformas (Java y .Net) para favorecer el atributo de calidad Interoperabilidad. | ***Responsable*:** Ingenium |
| ***Propósito:***  ( ) Reparación, actualizar, clarificar  ( X ) Obtener Información técnica  ( ) Obtener información de negocio  ( ) Otros: | |
| ***Propósito:***  Definir un esquema XML para estandarizar la comunicación entre el sistema central y los sistemas externos.  Comunicar sistemas desarrollados bajos distintas tecnologías. | |
| ***Descripción del experimento:***   * Definir un esquema XML para la comunicación entre el sistema central y los sistemas externos. * Construir un Servicio en plataforma .Net que simula el servicio del sistema externo policía. * Desde el sistema central consumir el servicio suministrado por el sistema externo (policía). Dicha comunicación se realizará haciendo uso del esquema XML definido inicialmente. | |
| ***Artefactos Creados:***  ExternalSystem ->Proyecto desarrollado en Microsoft.Net que simulará un sistema externo (policía). Este sistema publicará un WEB service que será consumido por el sistema central.  WEBServiceNotificator -> Módulo en el sistema central que hace llamado al servicio del sistema externo. | |
| ***Criterio de terminación:***  El sistema central debe poder comunicarse con el sistema externo, enviar una notificación y recibir la respuesta del sistema externo haciendo uso del esquema definido inicialmente. | |
| ***Recursos Requeridos:***  Recurso humano -> Se necesita el trabajo de 2 personas con conocimientos en java, JEE, JSF, Microsoft.Net y web services.  Recursos técnicos -> NetBeans IDE 7.0, Microsoft VS.Net 2008. | |
| ***Duración estimada:***  10 horas | |

Tabla 2. Resultados del Experimento A1

|  |
| --- |
| **Resultados del Experimento** |
| ***Definición de esquema xml:***  ***Esquema 1:*** *Este esquema xml permite al sistema central informar a un sistema externo acerca de la generación de un evento que solicita la atención por parte del sistema externo.*  *Homeid: Identificador de la casa que generó el evento.*  *EventType: Tipo de evento generado en la casa*  *Time: Fecha y hora de generación del evento*  *ContactName: Nombre del contacto asociado a la casa (dueño y/o habitante)*  *CantactPhone: Teléfono donde se puede ubicar al dueño y/o habitante de la casa.*  <?xml version="1.0" encoding="utf-8"?>  <xs:schema id="XMLNotificationSchema"  targetNamespace="http://tempuri.org/XMLNotificationSchema.xsd"  elementFormDefault="qualified"  xmlns="http://tempuri.org/XMLNotificationSchema.xsd"  xmlns:mstns="http://tempuri.org/XMLNotificationSchema.xsd"  xmlns:xs="http://www.w3.org/2001/XMLSchema"  >  <xs:element name="notification">  <xs:complexType>  <xs:sequence>  <xs:element name="homeid" />  <xs:element name="eventtype" ></xs:element>  <xs:element name="time"></xs:element>  <xs:element name="contactname"></xs:element>  <xs:element name="contactphone"></xs:element>  <xs:element name="address"></xs:element>  <xs:element name="city"></xs:element>  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:schema>  ***Esquema 2:*** *Con este esquema xml, el sistema externo genera una respuesta para el sistema central, respuesta que confirma que la notificación se realizó correctamente.*  *Homeid: Identificador de la casa de la cual se recibió la notificación.*  *EventType: Evento que fue notificado.*  *Message: Mensaje de respuesta generado por el sistema externo, donde informa el tiempo aproximado de llegada al lugar de los hechos.*  <?xml version="1.0" encoding="utf-8"?>  <xs:schema id="XMLResponseSchema"  targetNamespace="http://tempuri.org/XMLResponseSchema.xsd"  elementFormDefault="qualified"  xmlns="http://tempuri.org/XMLResponseSchema.xsd"  xmlns:mstns="http://tempuri.org/XMLResponseSchema.xsd"  xmlns:xs="http://www.w3.org/2001/XMLSchema"  >  <xs:element name="response">  <xs:complexType>  <xs:sequence>  <xs:element name="homeid" />  <xs:element name="eventtype" ></xs:element>  <xs:element name="message"></xs:element>  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:schema>  ***WSDL Asociado:*** *A continuación se presenta el wsdl publicado por el sistema externo que permite la comunicación entre los dos sistemas.*  <?xml version="1.0" encoding="utf-8" ?>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:definitions xmlns:soap="**http://schemas.xmlsoap.org/wsdl/soap/**" xmlns:tm="**http://microsoft.com/wsdl/mime/textMatching/**" xmlns:soapenc="**http://schemas.xmlsoap.org/soap/encoding/**" xmlns:mime="**http://schemas.xmlsoap.org/wsdl/mime/**" xmlns:tns="**http://tempuri.org/**" xmlns:s="**http://www.w3.org/2001/XMLSchema**" xmlns:soap12="**http://schemas.xmlsoap.org/wsdl/soap12/**" xmlns:http="**http://schemas.xmlsoap.org/wsdl/http/**" targetNamespace="**http://tempuri.org/**" xmlns:wsdl="**http://schemas.xmlsoap.org/wsdl/**">  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:types>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:schema elementFormDefault="**qualified**" targetNamespace="**http://tempuri.org/**">  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:element name="**HelloWorld**">    <s:complexType />    </s:element>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:element name="**HelloWorldResponse**">  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:complexType>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:sequence>    <s:element minOccurs="**0**" maxOccurs="**1**" name="**HelloWorldResult**" type="**s:string**" />    </s:sequence>    </s:complexType>    </s:element>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:element name="**NotifyEvent**">  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:complexType>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:sequence>    <s:element minOccurs="**0**" maxOccurs="**1**" name="**xmlEventNotification**" type="**s:string**" />    </s:sequence>    </s:complexType>    </s:element>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:element name="**NotifyEventResponse**">  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:complexType>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <s:sequence>    <s:element minOccurs="**0**" maxOccurs="**1**" name="**NotifyEventResult**" type="**s:string**" />    </s:sequence>    </s:complexType>    </s:element>    </s:schema>    </wsdl:types>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:message name="**HelloWorldSoapIn**">    <wsdl:part name="**parameters**" element="**tns:HelloWorld**" />    </wsdl:message>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:message name="**HelloWorldSoapOut**">    <wsdl:part name="**parameters**" element="**tns:HelloWorldResponse**" />    </wsdl:message>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:message name="**NotifyEventSoapIn**">    <wsdl:part name="**parameters**" element="**tns:NotifyEvent**" />    </wsdl:message>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:message name="**NotifyEventSoapOut**">    <wsdl:part name="**parameters**" element="**tns:NotifyEventResponse**" />    </wsdl:message>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:portType name="**PoliceServiceSoap**">  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:operation name="**HelloWorld**">    <wsdl:input message="**tns:HelloWorldSoapIn**" />    <wsdl:output message="**tns:HelloWorldSoapOut**" />    </wsdl:operation>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:operation name="**NotifyEvent**">    <wsdl:input message="**tns:NotifyEventSoapIn**" />    <wsdl:output message="**tns:NotifyEventSoapOut**" />    </wsdl:operation>    </wsdl:portType>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:binding name="**PoliceServiceSoap**" type="**tns:PoliceServiceSoap**">    <soap:binding transport="**http://schemas.xmlsoap.org/soap/http**" />  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:operation name="**HelloWorld**">    <soap:operation soapAction="**http://tempuri.org/HelloWorld**" style="**document**" />  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:input>    <soap:body use="**literal**" />    </wsdl:input>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:output>    <soap:body use="**literal**" />    </wsdl:output>    </wsdl:operation>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:operation name="**NotifyEvent**">    <soap:operation soapAction="**http://tempuri.org/NotifyEvent**" style="**document**" />  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:input>    <soap:body use="**literal**" />    </wsdl:input>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:output>    <soap:body use="**literal**" />    </wsdl:output>    </wsdl:operation>    </wsdl:binding>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:binding name="**PoliceServiceSoap12**" type="**tns:PoliceServiceSoap**">    <soap12:binding transport="**http://schemas.xmlsoap.org/soap/http**" />  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:operation name="**HelloWorld**">    <soap12:operation soapAction="**http://tempuri.org/HelloWorld**" style="**document**" />  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:input>    <soap12:body use="**literal**" />    </wsdl:input>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:output>    <soap12:body use="**literal**" />    </wsdl:output>    </wsdl:operation>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:operation name="**NotifyEvent**">    <soap12:operation soapAction="**http://tempuri.org/NotifyEvent**" style="**document**" />  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:input>    <soap12:body use="**literal**" />    </wsdl:input>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:output>    <soap12:body use="**literal**" />    </wsdl:output>    </wsdl:operation>    </wsdl:binding>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:service name="**PoliceService**">  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:port name="**PoliceServiceSoap**" binding="**tns:PoliceServiceSoap**">    <soap:address location="**http://localhost/ExternalSystemPolice/PoliceService.asmx**" />    </wsdl:port>  [**-**](http://localhost/ExternalSystemPolice/PoliceService.asmx?wsdl) <wsdl:port name="**PoliceServiceSoap12**" binding="**tns:PoliceServiceSoap12**">    <soap12:address location="**http://localhost/ExternalSystemPolice/PoliceService.asmx**" />    </wsdl:port>    </wsdl:service>    </wsdl:definitions>  El sistema central al recibir un evento consume el servicio Web invocando a “NotifyEvent” y pasándole como parámetro una cadena en formato XML cumpliendo con el esquema 1. El sistema externo recibe dicha notificación, genera un mensaje de respuesta cumpliendo con el esquema 2 y lo retorna al invocador (sistema central). Por último el sistema central, presenta el mensaje proveniente del sistema externo.  El sistema externo simula un tiempo de respuesta lento. Aquí se hace uso de comunicación asíncrona, para que el sistema central consuma el servicio web por cada evento que se genera. De esta manera se envían notificaciones a los sistemas externos sin tener que esperar la respuesta de dicho sistema (asíncrono). Esto favorece en gran medida el tiempo de desempeño del sistema central.  La definición de un esquema XML para la comunicación de sistemas, favorece la interoperabilidad. Sin importar la tecnología que ha implementado el sistema externo, la comunicación se hace de manera transparente. Y si llegado el caso se requiere adicionar un nuevo parámetro para la comunicación, basta con modificar el esquema de comunicación, pero el servicio no requeriría modificación alguna.  La comunicación desde el sistema central, se realiza a través de un proxy generado de acuerdo al wsdl que publica el sistema externo. En caso de que el sistema externo adicione un nuevo servicio y éste requiera ser consumido por el sistema central, el proxy debe ser modificado / actualizado. |
| ***Duración Real:***  10 horas |
| ***Recursos Reales:***  Recurso humano -> Se necesita el trabajo de 2 personas con conocimientos en java, JEE, JSF, Microsoft.Net y web services.  Recursos técnicos -> NetBeans IDE 7.0, Microsoft VS.Net 2008. |
| ***Recomendaciones:***  Desde el sistema central se debe implementar un mecanismo de generación del proxy dinámico. Esto debido a que el web service suministrado por el sistema externo puede cambiar su estructura y/o agregar nuevos servicios. |

1. **Experimento B – Comunicación Asíncrona – Envío de correo electrónico**

Tabla 3. Descripción del Experimento B

|  |  |
| --- | --- |
| **Descripción del Experimento** | |
| ***Título*:** Interoperabilidad comunicación asíncrona | ***ID*:**B |
| ***Descripción*:**  Se deben comunicar los eventos de manera asíncrona haciendo uso del correo electrónico. | ***Responsable*:** Ingenium |
| ***Propósito:***  ( ) Reparación, actualizar, clarificar  ( X ) Obtener Información técnica  ( X ) Obtener información de negocio  ( ) Otros: | |
| ***Propósito:***  Verificar y validar los artefactos desarrollados en la arquitectura planteada de manera que sea posible la interoperabilidad para la comunicación de los eventos a los clientes de manera asíncrona. | |
| ***Descripción del experimento:***  Verificar la arquitectura para garantizar la interoperabilidad del sistema, de manera que se pueda comunicar de forma asíncrona con los operadores de correo en internet, con el fin de informar vía correo electrónico un evento pre configurado por el cliente.  Se integra a los desarrollos del sistema central un nuevo módulo encargado de realizar la notificación a los clientes de los eventos detectados a través de correo electrónico. De manera que se validen las decisiones de diseño asociadas a la interoperabilidad del sistema, cumpliendo con las expectativas planteadas. | |
| ***Artefactos Creados:***  ASHCentralSystem / Email Notificator ->Es un módulo del sistema central en la capa de notificación encargado de enviar las notificaciones de manera asíncrona a través a los operadores de correo electrónico. | |
| ***Criterio de terminación:***  El sistema central recibe un evento y realiza el envió de un correo electrónico al cliente a través de un servidor de correo en internet, informando del evento que ha sucedido den su hogar u oficina. | |
| ***Recursos Requeridos:***  Recurso humano -> Se necesita el trabajo de 2 personas con conocimientos en java, estilos y tácticas de arquitectura, bases de datos, Threads, etc.  Recursos técnicos -> Base de datos Oracle XE, NetBeans IDE 7.0, Microsoft Visio, JUnit, Cuenta de correo electrónico. | |
| ***Duración estimada:***  10 horas | |

Tabla 4. Resultados del Experimento B

|  |
| --- |
| **Resultados del Experimento** |
| ***Resumen de los resultados:***  Se agregó a la capa de notificación dentro del sistema central un nuevo módulo encargado de las notificaciones a través de correo electrónico, no se debió modificar ninguna otra parte del sistema, se descubrió que se deben validar los casos en los que se debe enviar notificación al cliente, haciendo uso de esta herramienta.  Adicionalmente se definió un formato de mensaje que permitiera estandarizar las comunicaciones a los clientes y permitir el uso del módulo EmailNotificator para notificar cualquier tipo de evento.  El mensaje recibido por un cliente tiene el siguiente aspecto.    En el sistema central se presenta un patrón arquitectural por capas, y adicionalmente se tiene módulos con responsabilidades muy claras, permitiendo bajo acoplamiento y alta extensibilidad, adicionalmente permite favorecer la interoperabilidad. De esta manera se garantizó que se cumplieran con las expectativas de los interesados, de manera específica agregando el nuevo tipo de modificación al sistema. |
| ***Duración Real:***  12 horas |
| ***Recursos Reales:***  Recurso humano -> Se necesita el trabajo de 2 personas con conocimientos en java, estilos y tácticas de arquitectura, bases de datos, Threads, etc.  Recursos técnicos -> Base de datos Oracle XE, NetBeans IDE 7.0, Microsoft Visio, JUnit, Cuenta de correo electrónico. |
| ***Recomendaciones:***  Se recomienda verificar las restricciones de negocio para hacer uso de cada uno de los tipos de notificación en el momento indicado.  Se debe realizar la notificación por correo al cliente en un hilo de ejecución diferente al hilo de ejecución de notificación de los sistemas de emergencia, o se debe realizar la notificación después de hacer la notificación a los sistemas de emergencia para no ir en contra de las restricciones ya definidas y los demás atributos de calidad.  Se debe seguir haciendo uso de la plantilla definida, permitiendo integrar cualquier tipo de notificación al módulo ya creado. |

1. **Experimento C – Video llamada**

Tabla 6. Descripción del Experimento C

|  |  |
| --- | --- |
| **Descripción del Experimento** | |
| ***Título*:** Video llamada de emergencia | ***ID*:** C |
| ***Descripción*:** Este experimento busca conseguir realizar una video llamada de emergencia como respuesta a un evento | ***Responsable*:** Ingenium |
| ***Propósito:***  ( ) Reparación, actualizar, clarificar  ( ) Obtener Información técnica  ( ) Obtener información de negocio  ( X ) Otros: | |
| ***Propósito:***   * Notificar a los suscriptores sobre la detección de un evento para el cual han sido previamente suscritos * Iniciar automáticamente las video llamadas de las cámaras. * Reportar el estado de las llamadas | |
| ***Descripción del experimento:***  Hacer uso del API de Skype para lograr que nuestra aplicación genere una video llamada desde un equipo (PC) que se encuentre instalado en el domicilio y que cuente con una cámara Web disponible y correctamente configurada. También se incluyó la funcionalidad necesaria para que en caso de que la video llamada sea fallida, se envíe un correo de notificación al usuario suscrito, informándole sobre el evento ocurrido; por ejemplo en el caso de que no sea contestada, o que el usuario al que se esté llamando no se encuentre disponible. | |
| ***Artefactos Creados:***  Se ha modificado el proyecto ASHHomeModule para que incluya las funciones necesarias para la realización de la video llamada desde el computador que se encuentra instalado en el hogar, así colas funciones necesarias para el envío del correo electrónico informativo en caso de que la llamada sea fallida | |
| ***Criterio de terminación:***  El equipo instalado en el hogar debe ser capaz de generar la llamada a través de Skype automáticamente, e iniciar la transmisión de video en caso de que la llamada sea contestada o enviar un correo de notificación en caso contrario. | |
| ***Recursos Requeridos:***  Recurso humano -> Se necesita el trabajo de 2 personas con conocimientos generales sobre desarrollo de software para el API de Skype y específicamente en Java para el desarrollo de la aplicación  Recurso técnico -> NetBeans IDE 7.0, Referencia de la documentación de la API de Skype. Cuenta de Skype activa. Herramienta de consola para Skype (Tracer.exe) | |
| ***Duración estimada:***  8 horas | |

Tabla 7. Resultados del Experimento C

|  |
| --- |
| **Resultados del Experimento** |
| ***Resumen de los resultados:***  Se revisó la documentación de la API de Skype con el fin de encontrar las instrucciones y comandos necesarios para la generación de llamadas. Una vez verificadas las instrucciones necesarias se procedió a realizar las llamadas a través de la consola de Skype por medio de la herramienta **Tracer.exe.** Por medio del uso de esta herramienta se pudo determinar el funcionamiento de las llamadas de Skype.  Este es un ejemplo de la consola de Skype    Uno de los requisitos para poder realizar la llamada desde Skype es que el equipo desde donde se realizara la llamada cuente con una cuenta activa de Skype y que se encuentre logeado en la misma. Si se cumple este requisito se puede generar una llamada de voz. Si la llamada de voz es respondida, se genera un mensaje acerca del estado de la llamada y se devuelve un identificador único de la llamada. Con este identificador se puede proceder a realizar la modificación de la llamada, para nuestro caso, iniciar la transmisión de video. Por medio de esta experimentación se pudo determinar que no es posible iniciar la llamada directamente con video puesto que el video es una característica de una llamada de voz que solo puede ser habilitada una vez la llamada inicial sea exitosa.  Con estos resultados, se procedió a crear un proyecto de prueba en Java para la generación de la llamada desde la aplicación basándose en otros ejemplos los cuales usan un framework de Skype que encapsula las funcionalidades de la API de Skype para su uso en Java de una manera más amigable y entendible. Sobre este proyecto de prueba se realizó la experimentación necesaria para verificar que se podía generar correctamente la llamada e iniciar el video de la misma desde la aplicación. Igualmente se realizó la implementación necesaria para enviar un mensaje en el chat de Skype informando previamente al usuario que se realizara una video llamada para que el usuario la conteste.  Una vez lograda los objetivos de la generación de la video llamada, se procedió a incluir la codificación generada en el proyecto de prueba en el proyecto **ASHHomeModule** previamente creado para soportar las funcionalidades de llamada y video llamada en Skype. También se modificó el proyecto para que en caso de que la llamada sea fallida, se envíe un correo de notificación al usuario para que tome las medidas correspondientes según el evento que se registre. |
| ***Duración Real:***  12 horas |
| ***Recursos Reales:***  Recurso humano -> Se necesita el trabajo de 2 personas con conocimientos en java, sockets, estilos y tácticas de arquitectura, así como el análisis de la documentación del API de Skype  Recursos técnicos ->NetBeans IDE 7.0, Documentación de Skype, cuenta de Skype activa, cuenta de correo activa, consola de Skype, Framework de Skype, librerías de Skype. |
| ***Recomendaciones:***  Uno de los limitantes de usar una video llamada de Skype es que se debe tener un computador encendido con una cuenta de Skype activa y logeada antes de poder iniciar la video llamada, esto conlleva al hecho que si el equipo se encuentra fuera de línea (apagado o sin red o sin logear) la llamada no va ser exitosa, lo cual genera una de deficiencia pues cualquiera de los actores del sistema puede deshabilitarlo intencional o accidentalmente.  Se pueden conectar diferentes cámaras a un mismo sistema, pero estas deben contar con identificadores únicos para poder iniciar la transmisión de video desde una de ellas, la limitante es que solo se puede transmitir desde una de ellas en un mismo instante de tiempo, es decir si se generan alertas en diferentes áreas del domicilio solo se generara la llamada del primer evento con la cámara registrada a ese evento. El resto de los eventos no podrán generar la llamada porque la cuenta activa ya se encuentra ocupada. |

1. **Referencias**

* Documentación de la API de Skype

<http://developer.skype.com/resources/public_api_ref.zip>

* Herramienta de consola de Skype

<http://developer.skype.com/resources/Tracer.exe>

* Ejemplos de desarrollo

<https://github.com/taksan/skype-java-api>

<http://skype.sourceforge.jp/index.php?Skype%20API%20For%20Java%20%28English%29>