INTERFAZ DE REALIDAD AUMENTADA PARA LA INTERNET DE LAS COSAS

Omar Eduardo Guzmán Álvarez

ANEXOS

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Popayán
2016

ANEXO A. ANALISIS DE MODELOS Y METODOLOGIAS DE REQUERIMIENTOS.

MODELO DE PROTOTIPOS.

Es un modelo o proceso evolutivo a través de la creación de prototipos cada vez más refinados, es útil cuando no se tiene claridad en los requerimientos, ni certeza del funcionamiento de la solución [1], es un modelo genérico aplicable a cualquier otra filosofía de desarrollo. Su principal valor es la comunicación tanto con el cliente como con el equipo de desarrollo a lo largo del proceso [1], lo que lleva a obtener un diseño rápido, una buena gestión de recursos como el tiempo y a gestionar los riesgos por ejemplo en el aspecto tecnológico evitando construir soluciones que no resuelven las necesidades del cliente. Su efectividad también está dada por la gestión del cambio al ser un modelo que gestiona los requerimientos de manera ágil [29]. Sus etapas son:

1 **Plan Rápido**: Permite la identificación de las variables a considerar para el desarrollo, identificando y delimitando el problema.
2 **Diseño rápido:** Aquí se identifican los requerimientos, es la parte más importante de la metodología donde existe una realimentación extendida con el cliente, y es aquí donde se debe entender claramente lo que se desea del producto. Para esta etapa se implementan las Historias de Usuario.
3 **Construcción del Prototipo:** Es una parte técnica donde se documenta el desarrollo del prototipo para futuros mantenimientos, aquí van las interfaces, las funciones, y el control de flujo, el cómo se va a construir.
4 **Desarrollo, entrega y retroalimentación:** Se programa la parte técnica y se prueba el prototipo con el usuario final, de tal manera que este brinde sus objeciones y sugerencias para poder cumplir sus requerimientos.
5 **Comunicación:** Aquí se delimita nuevamente el producto de acuerdo a las capacidades y los recursos del programador, llegando a un acuerdo con el usuario final para retomar de nuevo un diseño rápido y la creación de un nuevo prototipo más refinado y con mayor calidad. Uno de los factores críticos en esta etapa es una mala interpretación de las necesidades y objetivos tanto en el usuario como el desarrollador.
6 **Entrega del desarrollo final:** Cuando finalmente se tiene un prototipo que se resuelve los requerimientos del usuario, se entrega el producto bien documentado y con un estimado para su mantenimiento.

Como desventajas se tiene que el usuario se hace altas expectativas acerca del producto y toma el prototipo como producto final, dejando de lado aspectos como la calidad entre otros, escogiendo herramientas que brindan rapidez pero no robustez, sin embargo las ventajas de este método compensan con creces estas desventajas aportando eficiencia y eficacia para resolver un requerimiento que involucre HCI como en la AR, esto se logra estableciendo criterios como: El prototipo define los requerimientos, puede ser descartable, y después del desarrollo se debe invertir tiempo para garantizar la calidad del producto final.

Algunos autores hacen la consideración que el prototipo puede ser o no descartable, siendo que el primer sistema desarrollado sirve para clarificar e identificar mejor los requerimientos, la mayoría de veces es mejor descartarlo y volver a comenzar a crear un mejor prototipo desde ceros, pero una vez clarificados los requerimientos , no necesariamente el siguiente prototipo es descartable, e incluso puede ser evolutivo, la clave está en ser claros con el cliente y hacerle entender las reglas del modelo [1].

METODOLOGIAS DE CAPTURA DE REQUERIMIENTOS.

- **Casos de uso**

  Es una manera estilizada de presentar las funcionalidades de los requerimientos, describiendo las actividades que tienen lugar por parte de los usuarios para ejecutar determinado proceso, a través de roles definidos, donde sus actores generan eventos e interactúan entre ellos mismos o con el sistema [1]. Como primer paso para describir un caso de uso se definen los actores, pero a medida que se da el proceso de desarrollo se pueden descubrir nuevos. Luego se definen sus objetivos, sus funciones, y eventos para crear finalmente un diagrama de casos de uso. Aunque existen varios estilos de diagramas, uno de los más intuitivos es el estilo de UML con los elementos que se observan en la Figura 1.

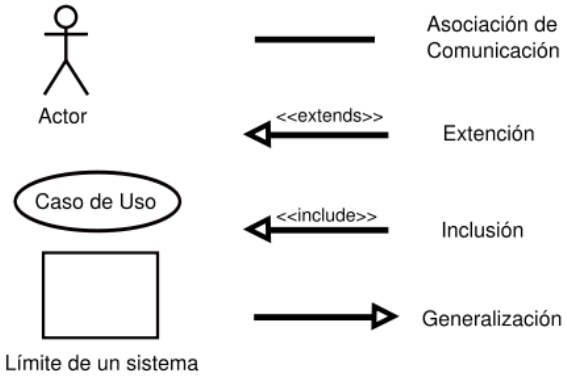

Figura 1. Casos de Uso

Tiene como ventaja el que permite observar las intenciones del usuario al comunicarse con el sistema, esto permite centrarse en los requerimientos del usuario y priorizarlos a en las primeras fases de desarrollo, sin embargo los casos de uso no son suficientes para establecer funcionalidad o requerimientos que no son funcionales. Para un desarrollo en AR la especificación de requerimientos mediante casos de uso seria rigurosa y generaría una etapa adicional respecto a la funcionalidad y la definición de los actores a lo largo del proceso.

- **Historias de usuario**

  Es una manera ágil e informal para la representación de un requisito de software escrito en una o dos frases utilizando lenguaje natural del usuario. Se caracteriza por ser breve, de una sola iteración, requiere conversación con el cliente para crear un plan. Este plan abarca los aspectos de la obtención de los objetivos, el análisis y la delimitación del proyecto.

Las historias de usuario son el registro de una conversación con el cliente para establecer las tareas a realizar y poder probar la solución desarrollada como respuesta a los requerimientos del cliente, se utilizan para priorizar los requerimientos, alcances y tiempo de realización [2]. Las historias de usuario se ajustan al paradigma de la construcción de prototipos que establece: escuchar al cliente, crear un prototipo y probarlo con el cliente para realizar las correcciones necesarias y refinar el prototipo.

Se justifica utilizar historias de usuario porque sigue la filosofía de una metodología ágil, potenciando la toma de decisiones y la cooperación por parte del equipo de trabajo, como también el trabajo en solitario, permite que la metodología utilizada sea evolutiva y altamente tolerante a riesgos, también permite evaluar requerimientos pequeños conteniendo la información verdaderamente necesaria para el desarrollo del proyecto [1].

- **Descripción de Tareas.**

  Es un método para adquirir requerimientos que describe tareas evitando separar al usuario de la maquina [3] a diferencia de los casos de uso. Al fundamentarse en las tareas, estas deben ser muy bien definidas, para esto se consideran una serie de ítems como el propósito de la tarea, la precondición o detonante (*trigger)*, una frecuencia de ocurrencia, y la medida crítica o donde la tarea puede colapsar el sistema, también puede definir sub tareas, y variantes de la misma. La descripción de tareas deja de lado aspectos como la eficiencia y la usabilidad para centrarse en la calidad, tiene cierto parecido con las historias de usuario pero con una secuencia definida y ordenada sin embargo esto representa un poco de rigurosidad, además que al definir tareas que el usuario y el sistema realizan juntos, se debe considerar y lidiar con este aspecto a lo largo del proceso de desarrollo.

HISTORIAS DE USUARIO

Historia de Usuario					
Identificación	**1**	Nombre	**Establecer contexto**		
Usuario					
Modificación de Historia Número			Iteración Asignada		
Prioridad en Negocio (Alta – Media – Baja)	**Alta**		Tipo de Actividad (Nueva – Arreglo – Cambio)	**Nueva**	
Riesgo en desarrollo (Alto – Medio – Bajo)	**Bajo**		Puntos estimados		
Desarrollador Encargado					
Descripción	**El usuario desea poder reconocer el escenario o contexto asociado a través de la aplicación mediante su dispositivo móvil, además de los recursos IoT disponibles.**				
Tareas	**-Establecer un mecanismo de identificación de la entidad de interés.** **-Establecer el contexto a partir de la identificación del escenario.** **-Detectar los objetos asociados al contexto.** **-Visualizar los recursos IoT (sensores, actuadores) asociados al contexto**				

	establecido en la interfaz de usuario.
Observaciones	• **Se parte de que el escenario sobre el cual se realizará la experiencia de AR ya está definido y por tanto existe un índice semántico que se puede consultar para realizar los procesos de contextualización en el área de la domótica.** • **Aunque el escenario inteligente se establece previamente, no están establecidos entre los objetivos la autentificación ni la personalización del usuario.** • **La seguridad de la aplicación no hace parte del desarrollo de este proyecto.** • **Se priorizará la gestión de la información de los recursos IoT, para luego realizar su montaje en una interfaz de realidad aumentada.** • **Fecha: enero 25 de 2016, Revisión: 31 de mayo de 2016**

Tabla 1. Historia de usuario 1.

Historia de Usuario				
Identificación	**2**	Nombre	**Consultar Datos**	
Usuario				
Modificación de Historia Número		Iteración Asignada		
Prioridad en Negocio (Alta – Media – Baja)		Tipo de Actividad (Nueva – Arreglo – Cambio)	**Nueva**	
Riesgo en desarrollo (Alto – Medio – Bajo)	**Alto**	Puntos estimados		
Desarrollador Encargado				
Descripción	**El usuario desea visualizar los metadatos del objeto inteligente cuando este se encuentre en su campo de visión lejano.**			
Tareas	• **Asignar una posición en el espacio tridimensional a la representación virtual del OI.** • **Realizar la consulta de los metadatos del OI al índice asociado.** • **Visualizar la información obtenida dentro del espacio tridimensional.**			
Observaciones	• **Fecha: enero 26 de 2016, Revisión: 31 de Mayo de 2016**			

Tabla 2.Historia de usuario 2.

Historia de Usuario				
Identificación	**3**	Nombre	**Consultar OI**	
Usuario				
Modificación de Historia Número		Iteración Asignada		
Prioridad en Negocio (Alta – Media – Baja)		Tipo de Actividad (Nueva – Arreglo – Cambio)	**Nueva**	
Riesgo en desarrollo (Alto – Medio – Bajo)	**Bajo**	Puntos estimados		
Desarrollador Encargado				

Descripción	**El usuario debe poder obtener información acerca de las propiedades y características del OI cuando este se encuentre en su campo de visión cercano.**
Tareas	• **Establecer la comunicación con cada objeto inteligente en el escenario.** • **Visualizar la información obtenida en la interfaz de usuario.**
Observaciones	• **Fecha: enero 26 de 2016, Revisión: 31 de mayo de 2016**

Tabla 3. Historia de usuario 3.

Historia de Usuario				
Identificación	**4**	Nombre	**Servicio Básico**	
Usuario				
Modificación de Historia Número		Iteración Asignada		
Prioridad en Negocio (Alta – Media – Baja)		Tipo de Actividad (Nueva – Arreglo – Cambio)	**Nueva**	
Riesgo en desarrollo (Alto – Medio – Bajo)	**Medio**	Puntos estimados		
Desarrollador Encargado				
Descripción	**El usuario debe poder activar o desactivar los servicios básicos de cada objeto inteligente al tenerlo en mi campo de visión cercano.**			
Tareas	• **Realizar la consulta de los metadatos de cada OI para presentar el estado del servicio básico.** • **Realizar la petición al OI de encender o apagar el servicio básico correspondiente al OI seleccionado.**			
Observaciones	• **El estado del servicio básico se visualiza continuamente en la representación virtual correspondiente durante la interacción con el usuario.** • **Existe la posibilidad del activar el servicio básico de cada OI a través de la creación de servicios de interacción.** • **Fecha: enero 26 de 2016, Revisión: 31 de mayo de 2016**			

Tabla 4. Historia de usuario 4.

Historia de Usuario				
Identificación	**5**	Nombre	**Servicios de Interacción entre OI**	
Usuario				
Modificación de Historia Número		Iteración Asignada		
Prioridad en Negocio (Alta – Media – Baja)		Tipo de Actividad (Nueva – Arreglo – Cambio)	**Nueva**	
Riesgo en desarrollo (Alto – Medio – Bajo)	**Alta**	Puntos estimados		
Desarrollador Encargado				
Descripción	**El usuario debe poder gestionar la interacción entre OI. Para esto define los eventos, las condiciones y las acciones que se deben ejecutar en el escenario. En el modelo definido los eventos corresponden a los sensores**			

	de un OI, las acciones corresponden a los actuadores de un OI y la condición la define el usuario a criterio propio, a esto se le ha llamado servicio de interacción o ECA (Evento-Condición-Acción).
Tareas	• **Crear la interacción grafica necesaria para definir un servicio de interacción y relacionarlo con una representación tridimensional.** • **A partir de los datos obtenidos crear el archivo ECA correspondiente en cada uno de los OI que interactúan.** • **Permitirle al usuario gestionar los servicios de interacción.**
Observaciones	• **Se debe tener una política para evitar los conflictos de ejecución de Servicios de interacción o ECA's, entre si y también con los servicios básicos de los OI. Dado que estas políticas no hacen parte de este proyecto, la única regla escogida es apagar el servicio básico correspondiente antes de encender un ECA para evitar conflictos, además el usuario deberá evitar ECAS conflictivos.** • **Fecha: enero 26 de 2016, Revisión: 31 de mayo de 2016**

Tabla 5. Historia de usuario 5.

ANEXO B. IMPLEMENTACION DE VUFORIA EN UNITY.

PTC Vuforia es un producto de Software de la Empresa PTC, que se utiliza para realizar desarrollos en realidad aumentada, su página:

https://www.vuforia.com/

describiendo a fondo el producto. Para realizar el registro de usuario en Vuforia y poder utilizar los paquetes para Android, IOS o Unity 3D utilice el enlace:

https://developer.vuforia.com/

descargando el paquete para desarrollo de su preferencia (ver figura 2). Para este desarrollo, se utilizara el paquete de Vuforia para Unity 3D, una vez descargado, se abre el proyecto de Unity 3D, y se instala el paquete de Vuforia.

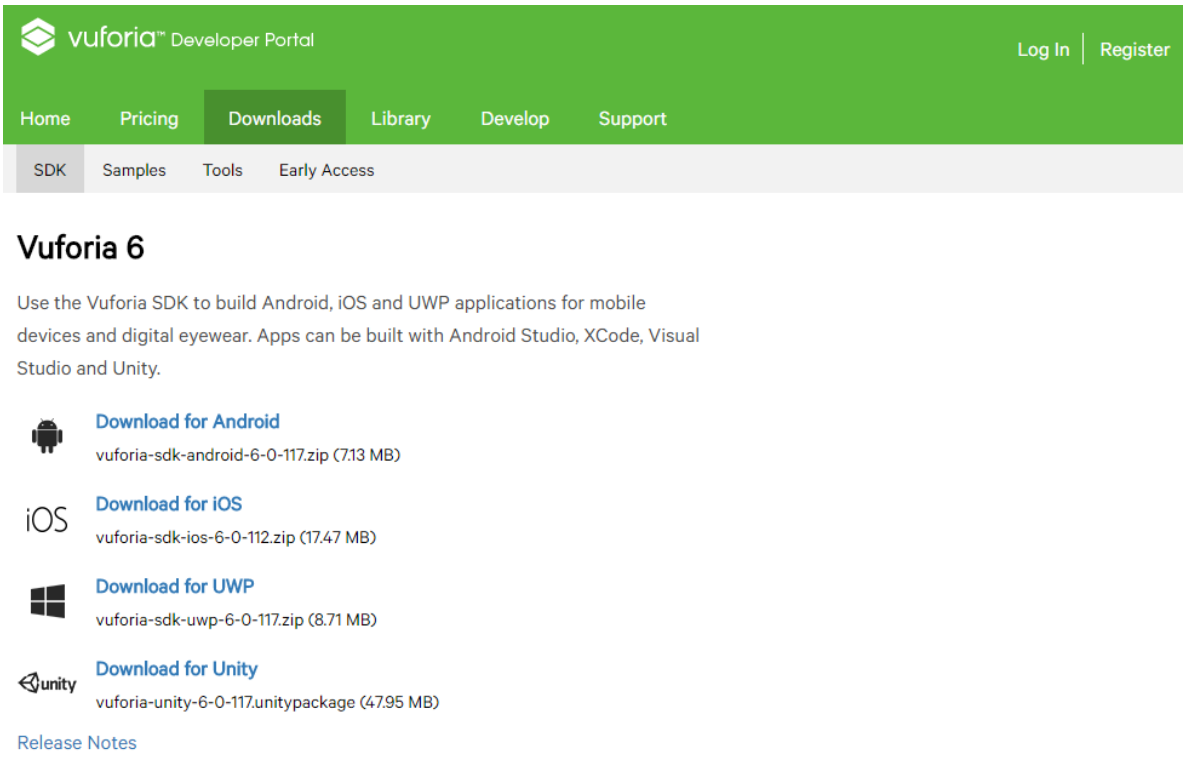

Figura 2. Paquetes de desarrollo de PTC Vuforia.

La forma más sencilla de utilizar Vuforia, es añadir al directorio del proyecto, la cámara de realidad aumentada ubicada en los prefabricados de la carpeta *Vuforia/Prefabs*, esto son: ***ARCamera*** y ***ImageTarget*** como se observa en la figura 3, para esto arrastre estos elementos desde la carpeta del proyecto hasta el árbol de Jerarquía llamado *Hierarchy* borrando previamente los elementos ***MainCamera*** y la ***DirectionalLight*** si la considera innecesaria.

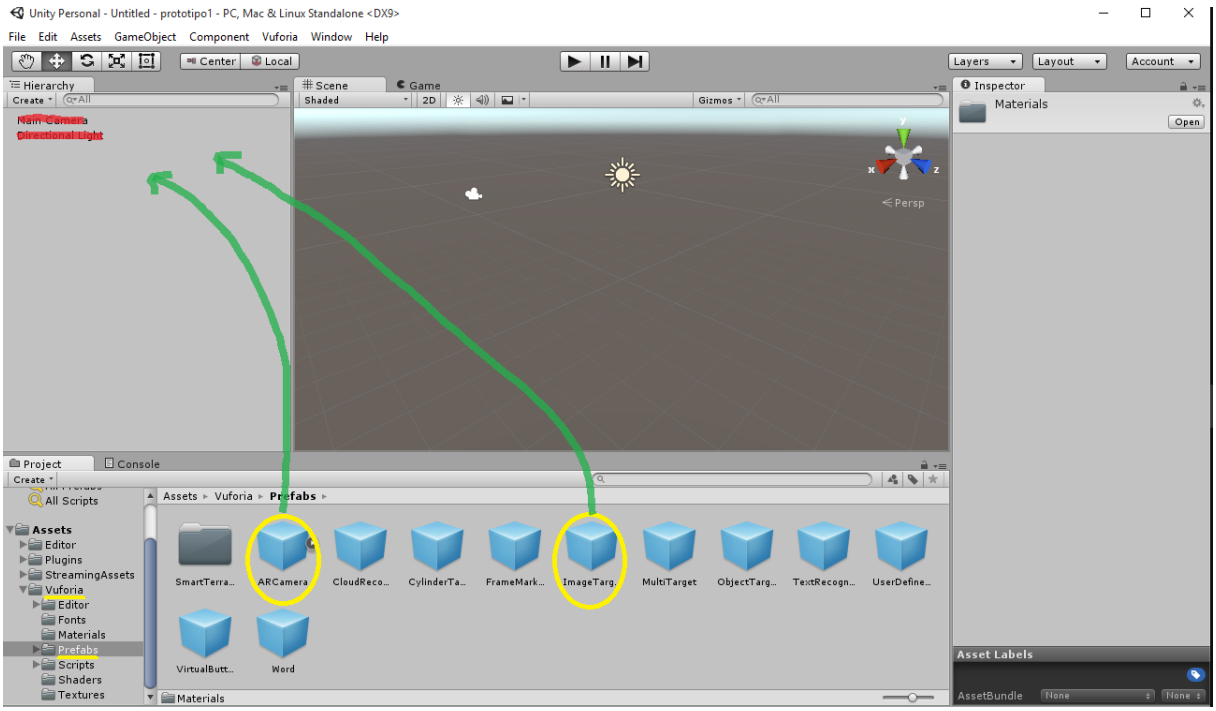

Figura 3. Prefabs básicos de Vuforia. Fuente Propia.

Luego en el portal de desarrollador de Vuforia se accede a la gestión de las licencias y los marcadores, comenzando por el Licence Manager como se observa en la siguiente figura.

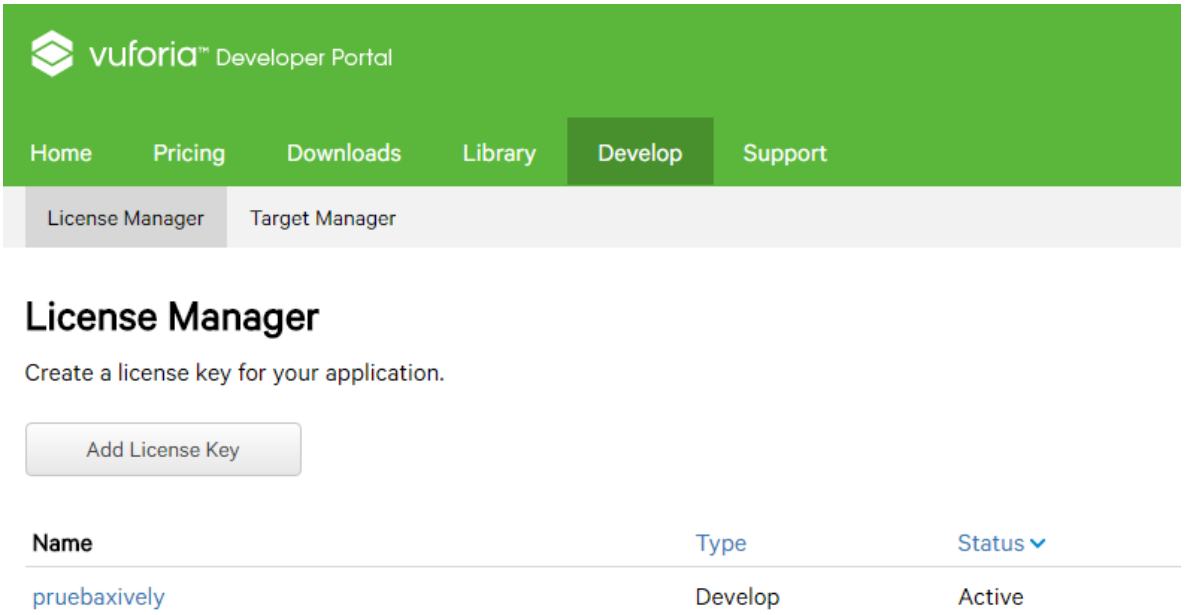

Figura 4. Portal de Desarrollador de Vuforia.

Aquí se tiene la licencia creada en el registro de Vuforia, se selecciona dicha licencia y se copia la llave de la misma, seguidamente se copia la llave de licencia en el inspector de la ***ARCamera*** como se ve a continuación en la figura 5.

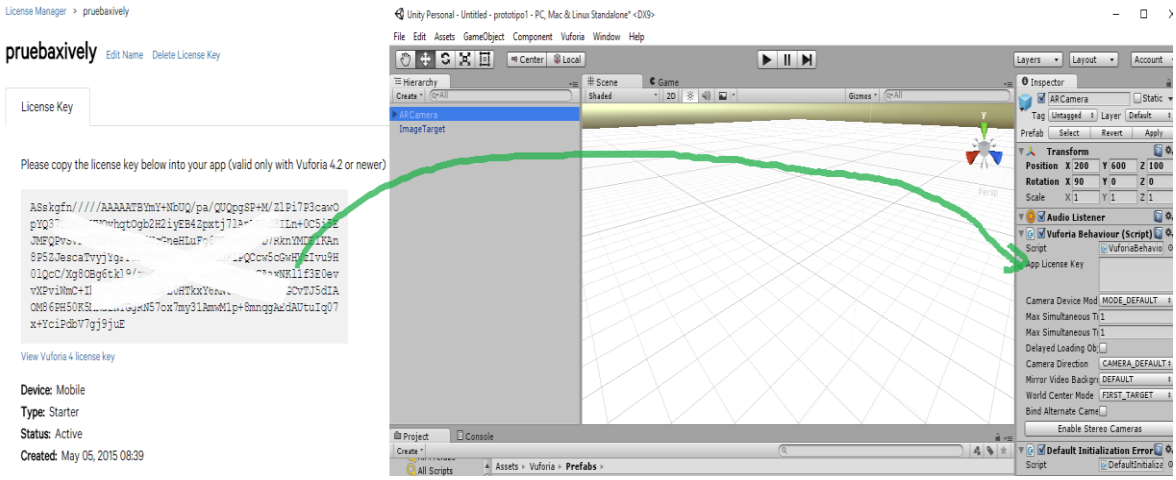

Figura 5. Introducción de la llave de licencia de Vuforia en Unity. Fuente Propia.

Al lado de la pestaña del ***Licence Manager*** se encuentra el ***Target Manager***, aquí se pueden subir los marcadores a reconocer, en esta ventana se crea una base de datos para los marcadores, se selecciona la base de datos creada se suben los marcadores, sean Imagen simple, código QR o imagen 3D, recordando que Vuforia le asigna a cada marcador un puntaje de la calidad del marcador respecto a su resolución y sus puntos de detección reconocibles. El ítem de *Width* indica un valor en mm con el cual Vuforia promediara la distancia del marcador hasta la cámara, un rango aceptable para comenzar esta entre 200 y 500mm (ver figura 6).

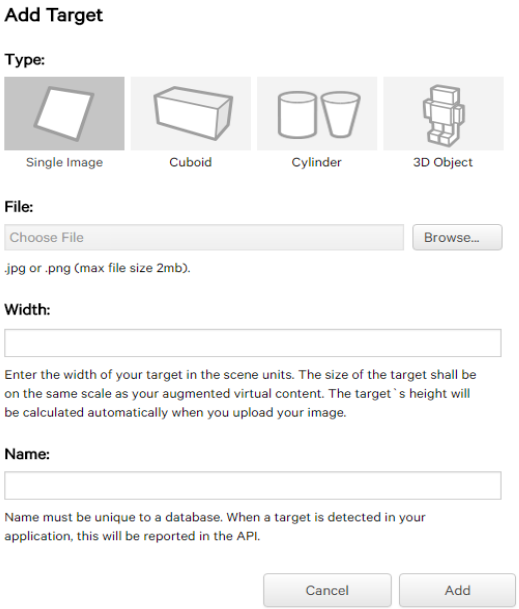

Figura 6. Carga de marcadores en Vuforia.

De nuevo en la interfaz de Unity 3D, se selecciona el inspector del ***ImageTarget*,** y se presiona el botón para seleccionar la base de datos de marcadores como se observa en la figura 7, el cual conduce de nuevo a la interfaz web de Vuforia.

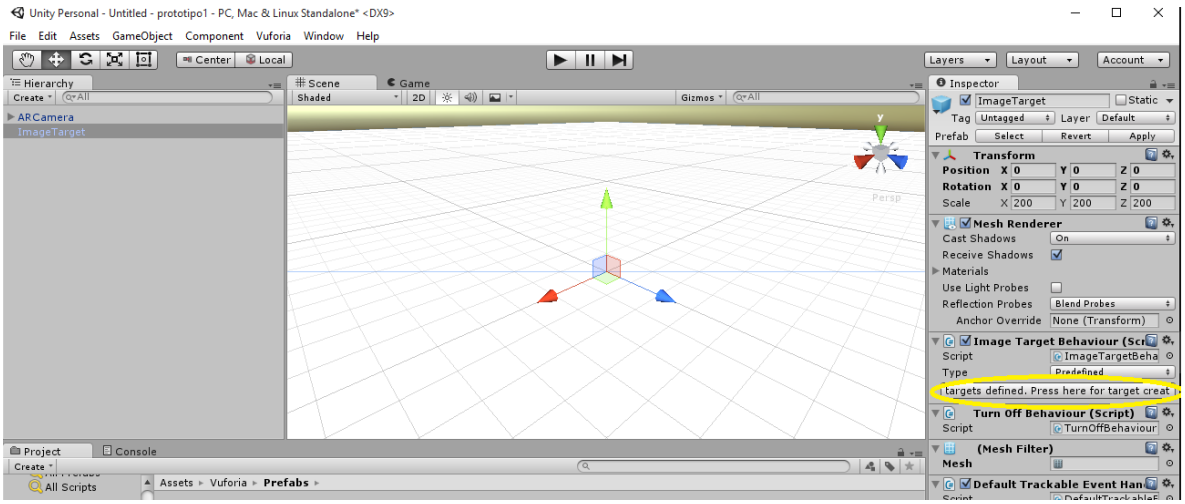

Figura 7. Carga de marcadores en Unity. Fuente Propia.

Se procede a importar la base de datos donde se encuentran los marcadores para el proyecto de Unity 3D, se descarga e importa el paquete generado, y en el mismo lugar del inspector habrá cambiado el menú dando la opción de escoger la base de datos y los marcadores previamente subidos como se observa en la figura 8.

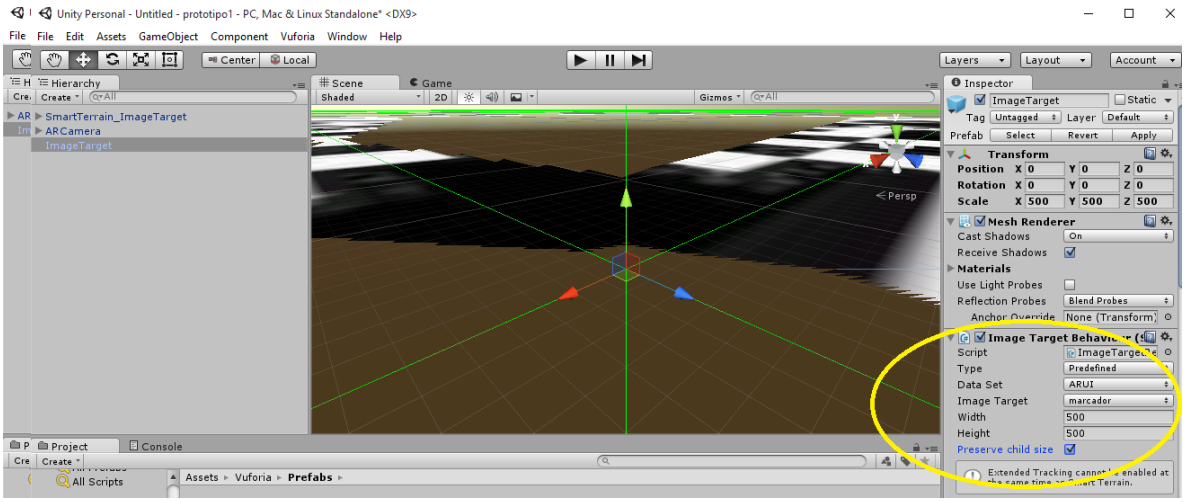

Figura 8. Marcadores en Unity. Fuente Propia.

Se escogen el *Data Set* y el *ImageTarget* en el inspector de GameObject ***ImageTarget*,** luego se selecciona la ***ARCamera*** y en su inspector se selecciona el *Data Set* escogido y se marca activo o *Active* como se ve en la figura 9.

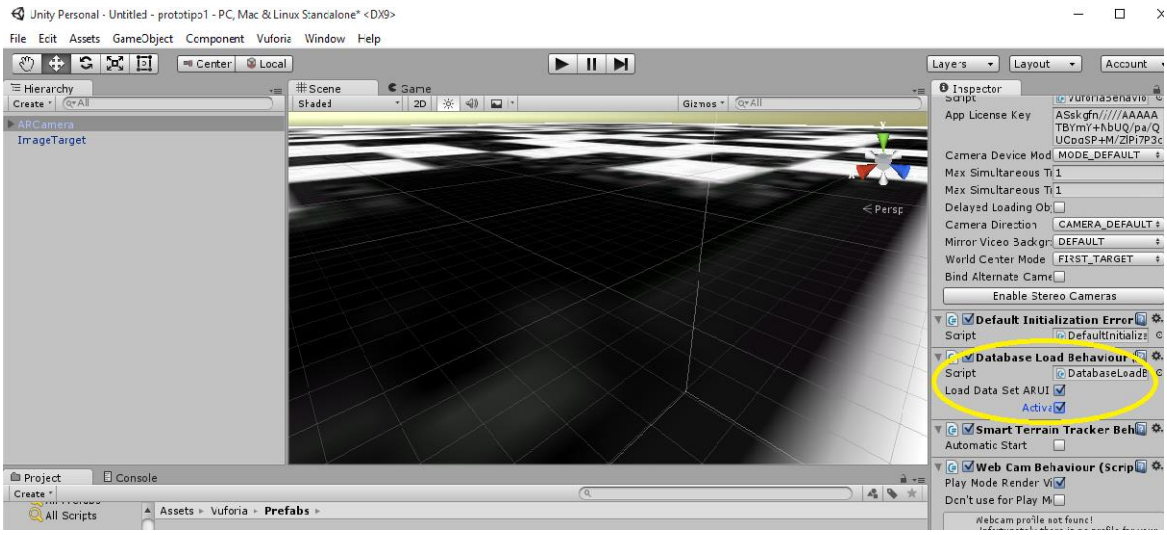

Figura 9. Activación de marcadores en Unity. Fuente Propia.

Dentro de la jerarquía del ***ImageTarget***, se pueden colocar los elementos que se van a relacionar con el marcador, por ejemplo una simple esfera, en el menú de Unity 3D se selecciona la pestaña *GameObjet* , ahí se encuentran desplegados una serie de objetos en 3D, se escoge alguno de ellos y se ubica dentro del ***ImageTarget*** *como se observa en la figura*, se escala y posiciona como se desee (ver figura 10).

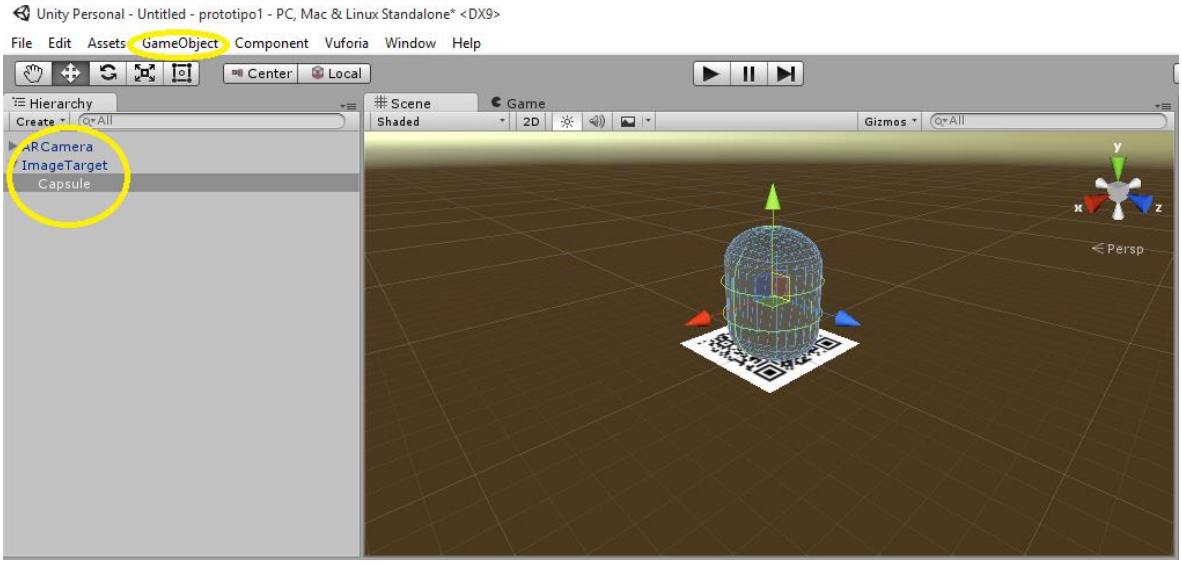

Figura 10. GameObject en el marcador objetivo. Fuente Propia.

Finalmente se ejecuta el proyecto con el botón de *Play* y se apunta el marcador a la cámara del computador como se ve en la figura 11, cuando la cámara del pc detecta el marcador a través de la aplicación, se dibujara el ***GameObject*** escogido.

Unity 3D permite observar los errores y advertencias, así como el flujo de la información en tiempo de ejecución, estas funcionalidades se pueden ver en la sección de las carpetas del

proyecto, ahí aparecen dos pestañas: *Project* y *Console*, en esta última se pueden observar los errores y advertencias que se generan al ejecutar un proyecto, además de los procesos generales.

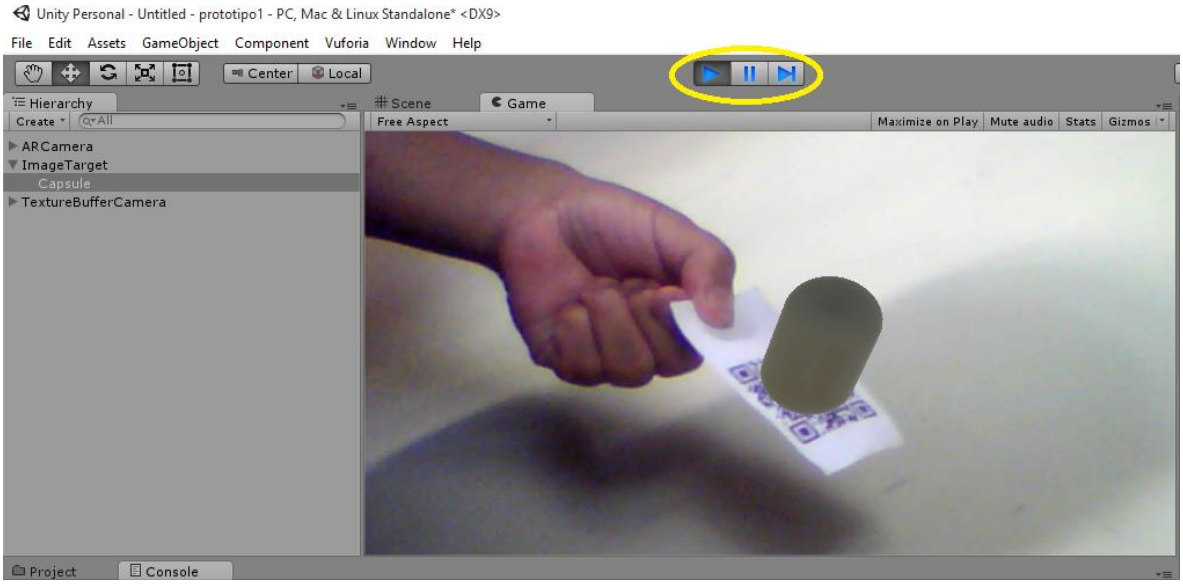

Figura 11. Marcador Detectado. Fuente Propia.

Unity permite la construcción de la aplicación para diferentes plataformas, desde la versión 5 de Unity 3D, para su instalación se descarga un asistente, el cual posee el paquete propio para cada plataforma, se hace necesario descargar cada paquete según las necesidades del desarrollador.

Para seleccionar la plataforma objetivo, una vez instalado su paquete respectivo, se predetermina la plataforma utilizando el botón "**switch plataform**", en el menú **File>Build Settings…,** ahí se deben incluir las escenas a implementar en la aplicación, abriendo cada escena de forma ordenada y agregándola a la aplicación, como se observa en la figura 12.

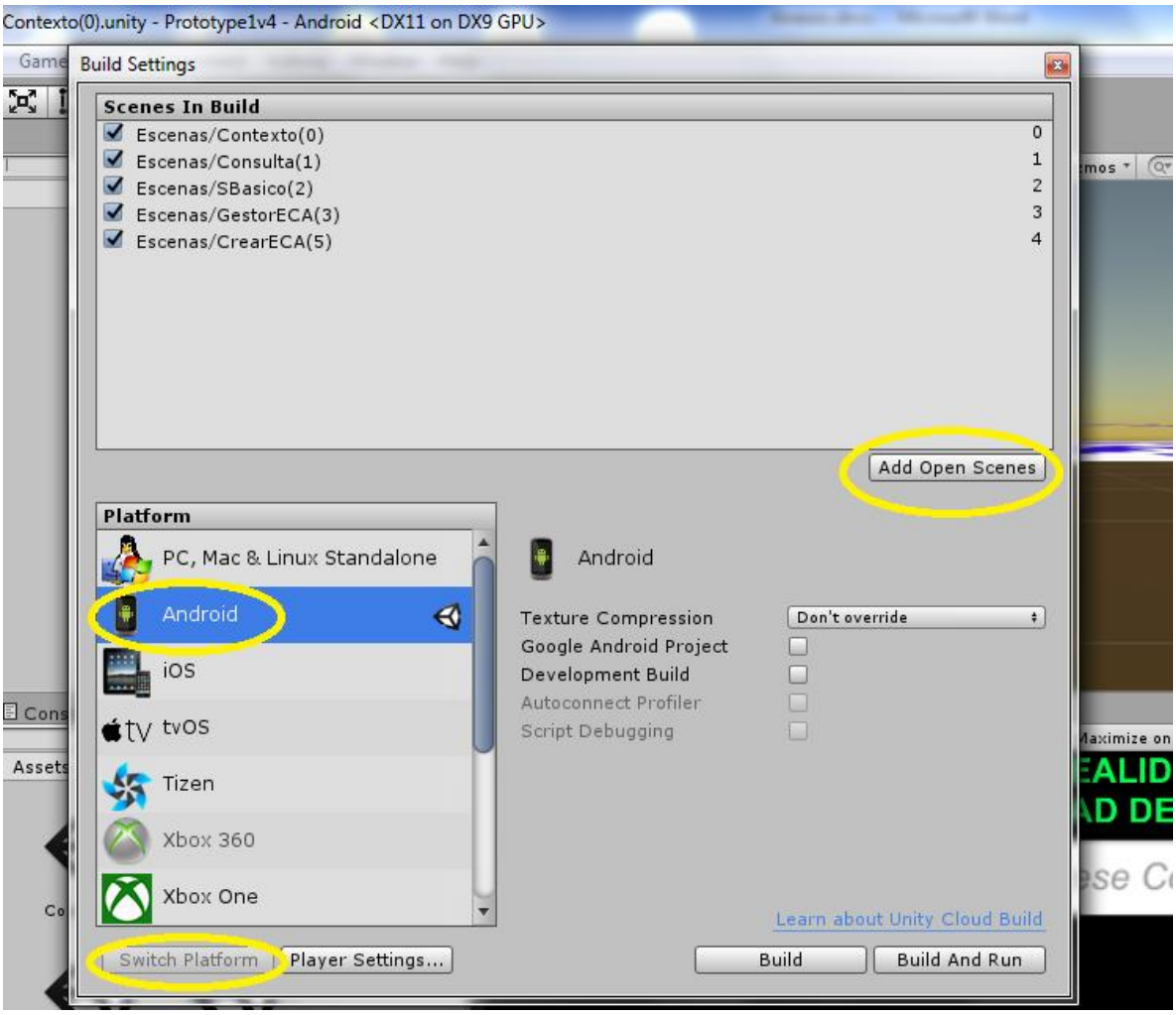

Figura 12. Selección de la plataforma de desarrollo y las escenas a implementar.

En particular para un desarrollo de Unity 3D hacia el Sistema Operativo Android, es muy importante establecer ciertos parámetros como el nombre de la aplicación , la compañía que lo desarrolla, la versión objetivo, entre otros, para esto se selecciona el botón de "**Player Settings**", donde aparecen todas estas opciones en un inspector, cabe notar que algunos de los diversos plugins y SDK's creados para Unity 3D poseen sus propios parámetros de empaquetado o *Bundle Settings,* lo cual hace necesario conocerlos y colocarlos debidamente en la asignación de la plataforma como se observa en la figura 13.

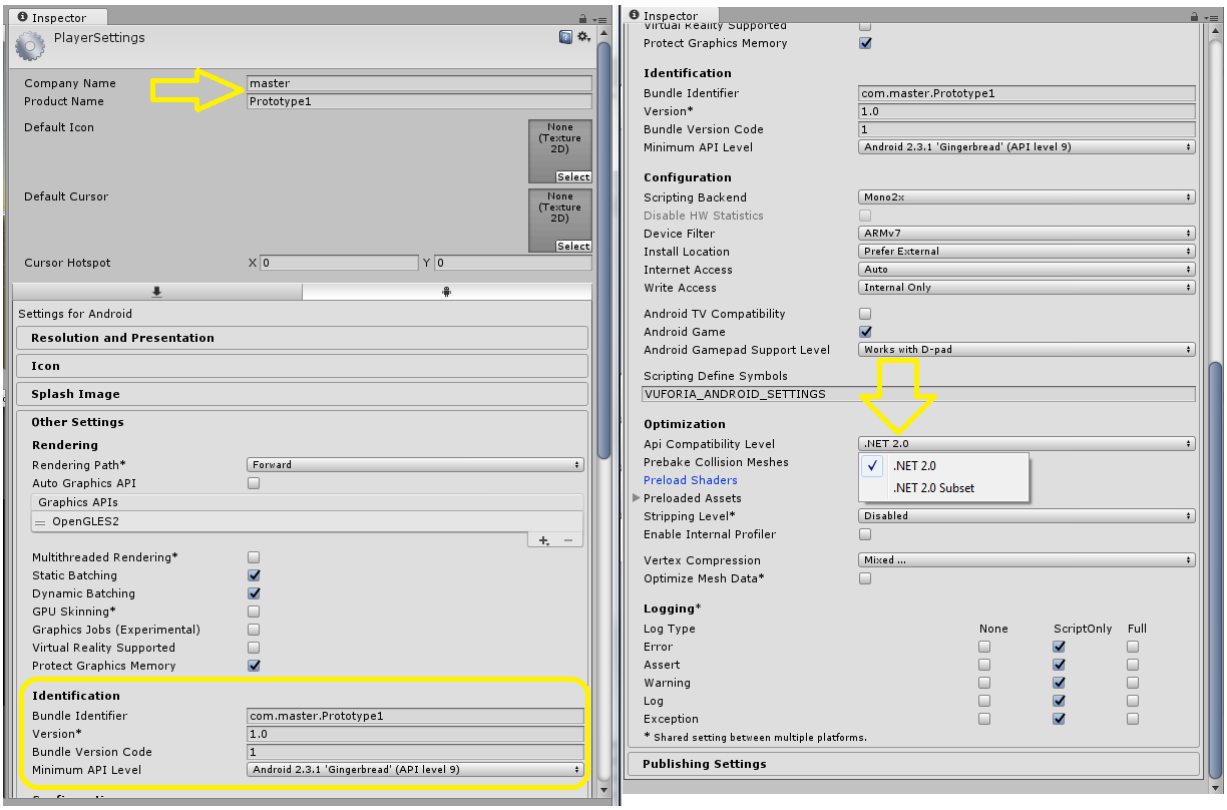

Figura 13. Player Settings de la plataforma Android en Unity 3D.

Una vez completado el anterior paso, con los parámetros de configuración correctos, se puede construir el archivo de instalación con extensión **".apk"**, para Android, mediante el botón "**Build**", o si se tiene conectado al computador algún dispositivo Android en modo Debugger, se puede utilizar el botón "**Build and Run**" de la ventana *Build Settings.*

ANEXO C. IMPLEMNETACION DE SERVICIOS WEB EN UNITY3D.

Un *Web Service* se puede referenciar en Unity 3D mediante los IDE integrados MonoDevelop, Xamarin. o mediante Microsoft Visual Studio de manera sencilla para el lenguaje C#, utilizando un cliente SOAP, para esto se crea un nuevo script en Unity, relacionado a un elemento y utilizarlo aunque no es la única manera.

Primero se crea un Script de C# en Unity sobre algún elemento de un proyecto dado, en el caso particular se tiene un *GameObject* tipo Capsula, ubicándose en inspector de este elemento, se hace click en el botón *Add Component*, creando un nuevo script, se escoge el lenguaje *C Sharp* dando click en el botón *Create and Add* como se observa en la figura 14.

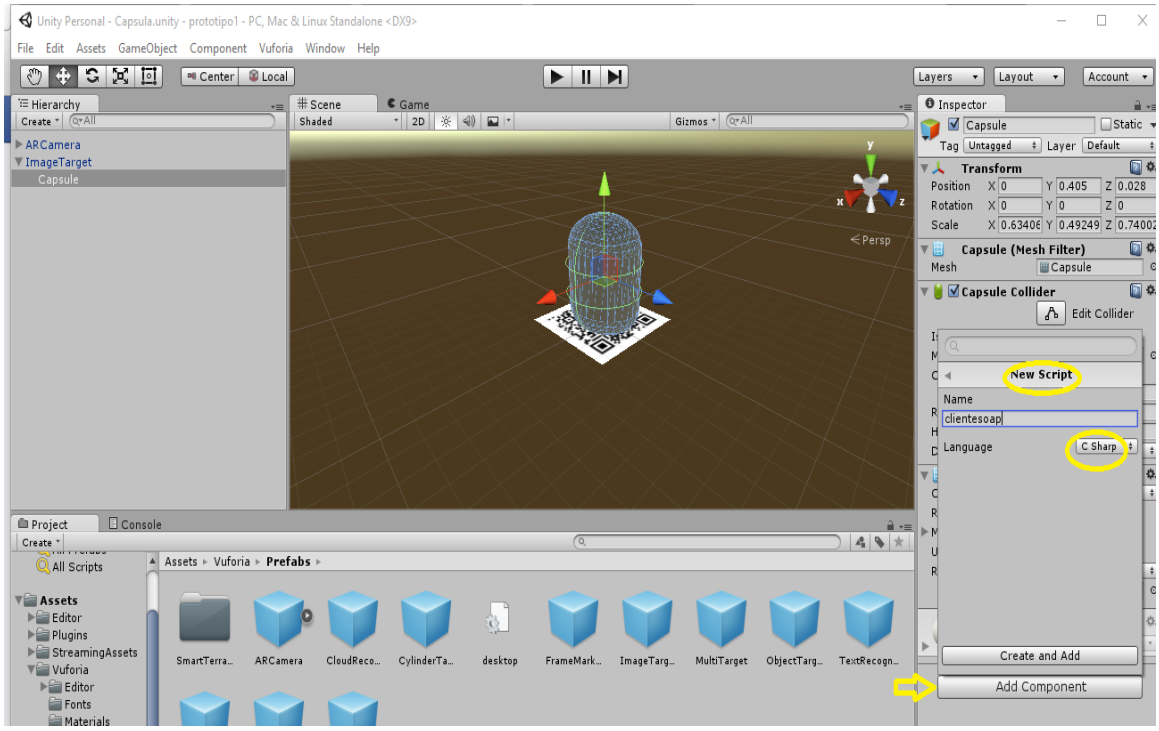

Figura 14. Creación de un Script en Unity sobre un elemento. Fuente Propia.

Para editar el Script, Unity utiliza el IDE Nativo llamado MonoDevelop, o también tiene la opción de edición con Microsoft Visual Studio o Xamarin, para MonoDevelop y Xamarin, se busca el script en la carpeta raíz del proyecto y se hace doble click sobre este, a continuación se abrirá el entorno de desarrollo por defecto como se observa en la figura 15.

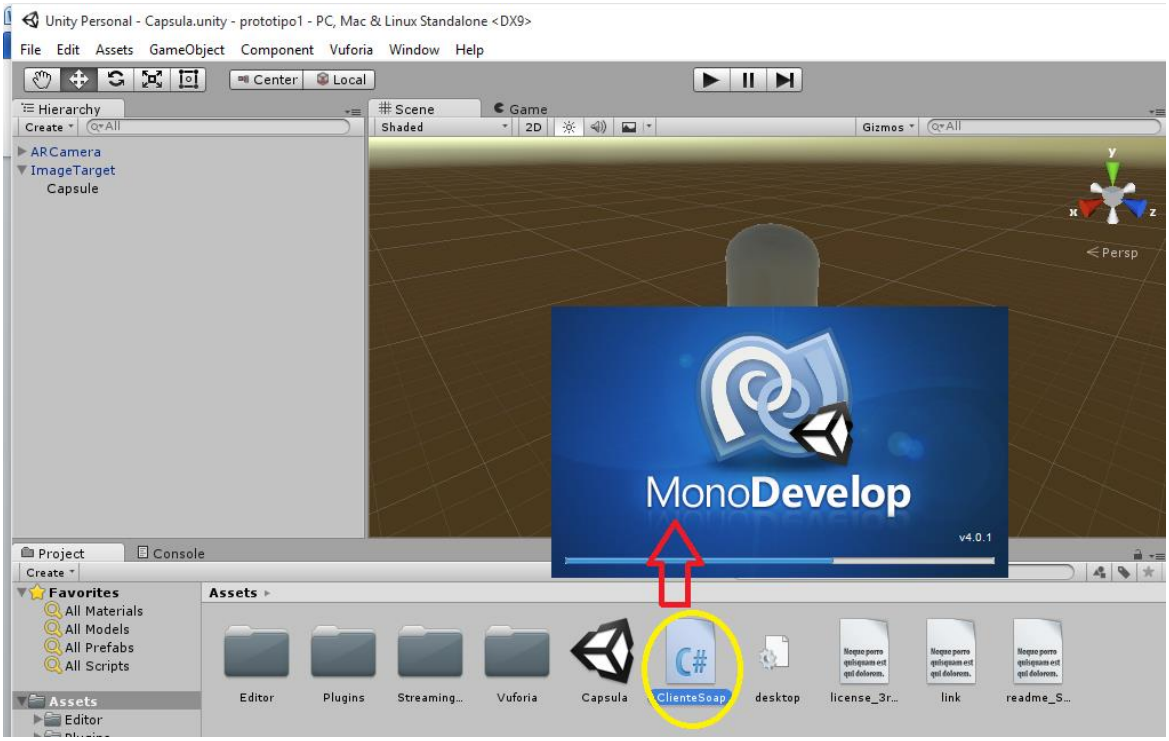

Figura 15. Abriendo un Script en el Editor de MonoDevelop. Fuente Propia.

En el entorno de desarrollo se crea la referencia al servicio web como se ve en la figura 16.

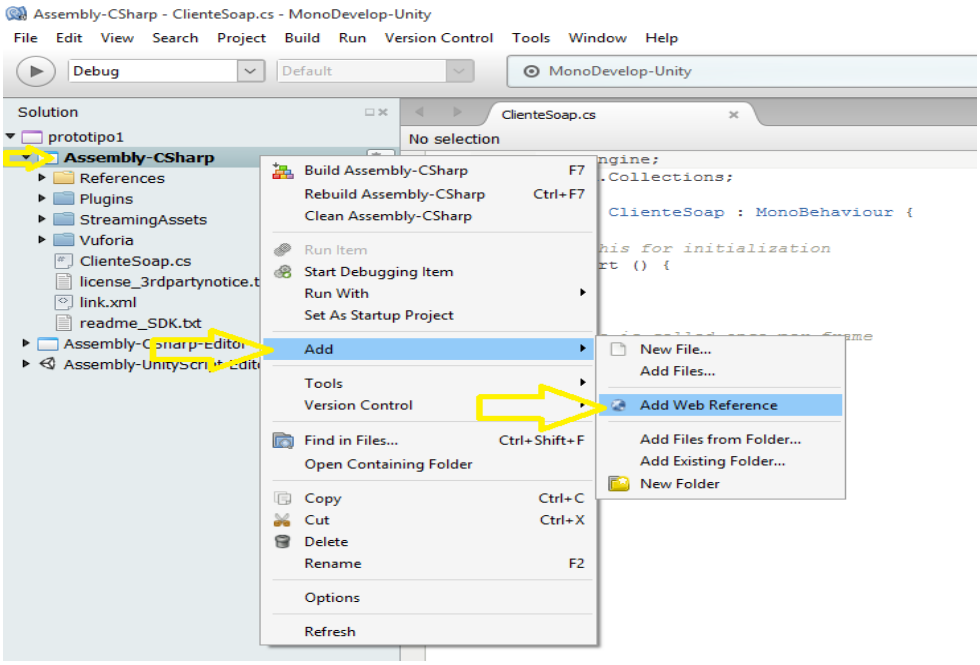

Figura 16. Creando la referencia web en el IDE.

A continuación se debe escribir la URL del servicio Web y se selecciona el tipo sea .NET o WCF y se le da un nombre, para este caso es un servicio .NET, utilizando el servicio que viene por defecto en MonoDevelop se tiene la URL:

http://www.w3schools.com/WebServices/TempConvert.asmx

Observe que es una extensión tipo .asmx, en un navegador se puede observar los servicios que presta y su descripción en XML como se ve en la figura 17.

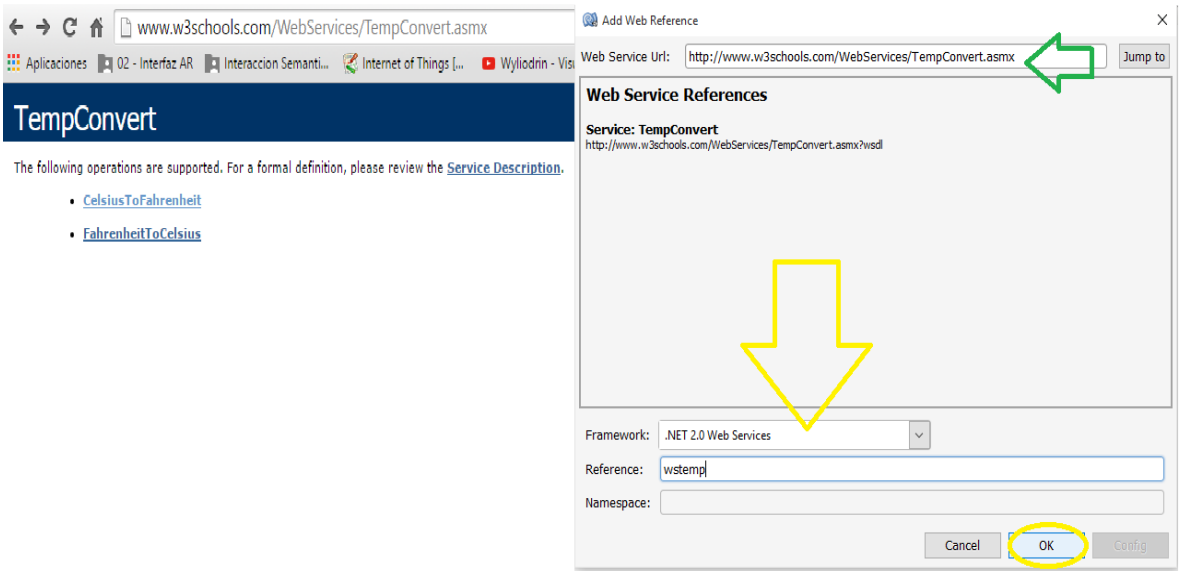

Figura 17.Características del Servicio Web.

Antes de crear el código para consumir el servicio, se deben incluir en la carpeta del proyecto algunas librerías necesarias que se encuentran en el directorio raíz de Unity, para el caso de 32 Bits la dirección seria:

C:\ Program Files\Unity\Editor\Data\Mono\lib\mono\2.0

Si no se incluyen estas librerías, el mismo compilador las va a requerir en su momento y no creara la referencia *Reference.cs* que tiene el mapeo de todos los servicios que se desean. Las librerías necesarias son:

System.Web.dll

System.Web.Services.dll
Y ocasionalmente requiere ***System.Data.dll***

Las cuales se copian en la carpeta del directorio raíz del proyecto en Unity 3D :

\Assets\Plugins

Automáticamente el entorno vuelve a compilar todo el proyecto creando la referencia como se observa en la figura 18. Algunos servicios requieren otras librerías o manejan variables que el entorno no soporta, pero observando los errores que arroja el compilador o la consola de Unity, es sencillo solucionar dichas fallas.

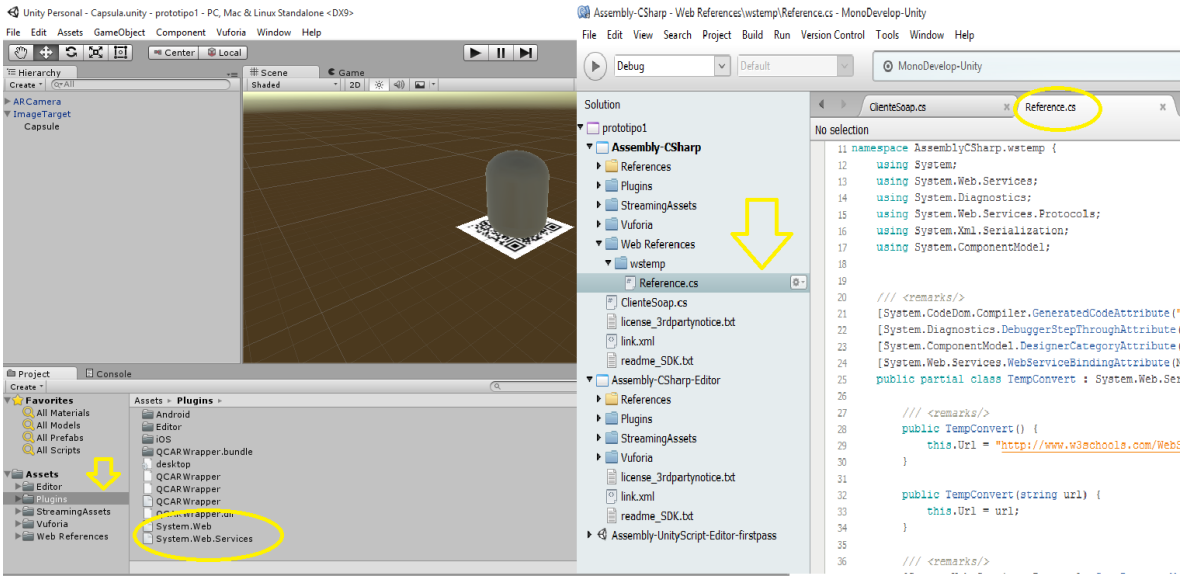

Figura 18. Referencia web creada.

También es necesario indicarle a Unity que el *framework* a trabajar, para este caso se escoge entre *Microsoft .NET* y *.NET Subset*, entonces en el entorno de Unity en el menú *File* se selecciona *Build Settings*, ahí se da click en el botón de *Player Settings* y en el Inspector se selecciona *.NET 2.0* como el *framework* a trabajar si así lo requiere la aplicación (ver figura 19).

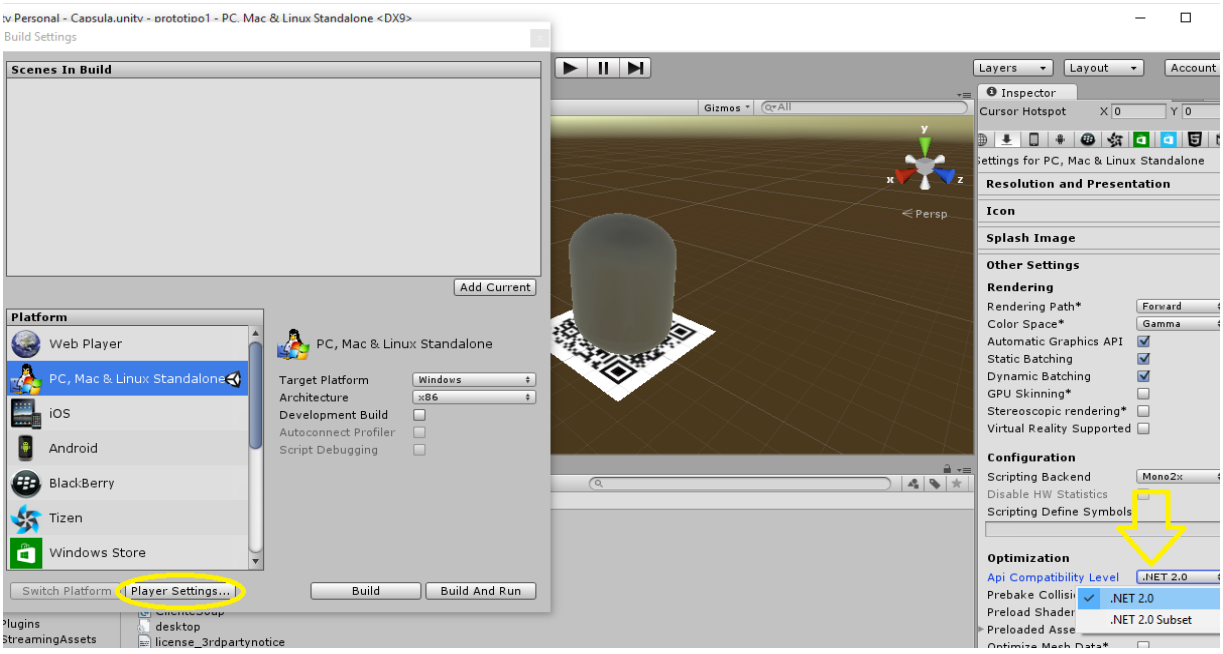

Figura 19. Selección del Framework de Microsoft .NET.

Basándose en la referencia, se crea el código necesario en el Script cliente SOAP, teniendo en cuenta los métodos nativos de Unity, *Start() y Update()*, ya que cada uno actúa de manera diferente en la interfaz.

Analizando el código creado se observa que al pasarle la cadena “15” como argumento a la función *CelsiusToFahrenheit(String)* , se obtiene como resultado la cadena “59”, tal cual lo que retorna el servicio Web al invocar dicha función, evidentemente 15 ºC son equivalentes a 59ºF. Observe en la figura 20 el código de la implementación del servicio web.

```
using UnityEngine;
using System.Collections;
using AssemblyCSharp.wstemp;  //Libreria del servicio Web

public class ClienteSoap : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        //Creacion del Objeto de la clase del Servicio Web cuyo nombre es TempConvert
        TempConvert wst = new TempConvert();
        //La funcion CelsiusToFahrenheit(String) retorna un valor tipo String
        string f = wst.CelsiusToFahrenheit("15");
        print(f);
    }
}
```

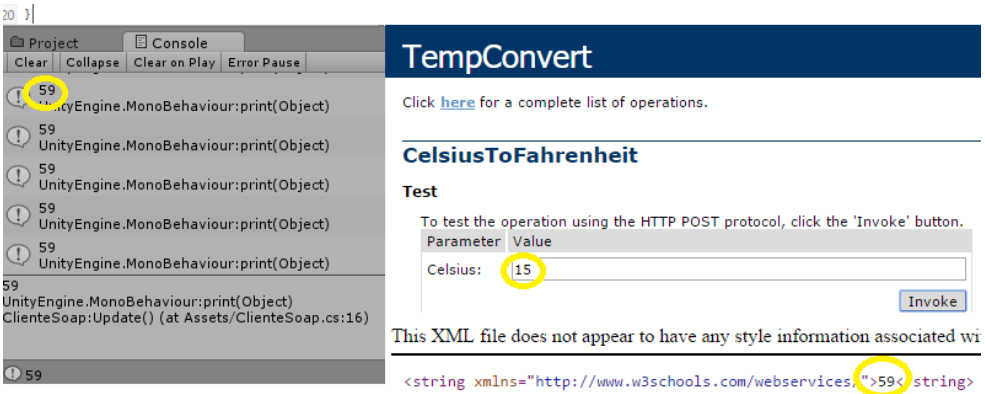

Figura 20. Implementación del servicio web en Unity.

ANEXO D. PROGRAMAS EN C#.

A continuación se describen los códigos creados para las funcionalidades de la interfaz de realidad aumentada para la IoT, teniendo en cuenta que Unity ofrece a C Sharp - C# como alternativa de código para la creación de scripts. La anatomía básica de un script en Unity se puede ver en el siguiente enlace:

https://docs.unity3d.com/es/current/Manual/CreatingAndUsingScripts.html

CÓDIGOS DE LAS FUNCIONALIDADES DE LOS MARCADORES DE VUFORIA EN UNITY.

Vuforia ofrece un paquete para Unity, donde tiene una serie de objetos prefabricados para las funcionalidades de realidad aumentada, cada uno con sus programas internos, donde algunos no son jerárquicos, es decir no tienen prioridad sobre los scripts que genera el usuario, esto permite que se capturen ciertos eventos como la detección de marcadores en la escena ejecutando ciertas acciones. A continuación están los códigos relacionados a las funciones que se relacionan con los eventos de detectar o perder un marcador en la escena en ejecución.

- ***Tracking.cs***

```
using UnityEngine;
using System.Collections;
using Vuforia;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Tracking : MonoBehaviour,
ITrackableEventHandler
{
    private TrackableBehaviour mTrackableBehaviour;
    bool ingreso = false;          //Variable que controla la deteccion del marcador
    public string contexto;        //Variable donde se almacena la palabra de la enti
dad de interes o consulta
    public Slider slider;          //Componente deslizador del radio de accion
    public GameObject deslizador;  //Objeto deslizador
    public GameObject input;       //Objeto GPS
    public GameObject HUD;         //Objeto HUD
    public Text texto2;            //Texto del deslizador
    public AudioSource open;
    public AudioSource close;

    void Start()
    {
        Input.location.Start ();    //Encendido del GPS
        mTrackableBehaviour = GetComponent<TrackableBehaviour>();
        if (mTrackableBehaviour)
        {
            mTrackableBehaviour.RegisterTrackableEventHandler(this);
        }
    }

    public void OnTrackableStateChanged(TrackableBehaviour.Status previousStatus,Trac
kableBehaviour.Status newStatus)
    {
```

```
        contexto=mTrackableBehaviour.TrackableName;
        if (newStatus == TrackableBehaviour.Status.DETECTED ||
            newStatus == TrackableBehaviour.Status.TRACKED ||
            newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
        {
            ingreso = true;
            open.GetComponentInChildren<AudioSource>().Play();
            print("se encontro un marcadooooooooor: " +contexto);
            PlayerPrefs.SetString("Context",contexto);    //Guardando el nombre de la
 entidad de interes

            Input.location.Start ();
            float latitud=Input.location.lastData.latitude;
            float longuitud=Input.location.lastData.longitude;
            float radio = slider.value;

            //Guardando los datos de la consulta
            PlayerPrefs.SetFloat("Lat",latitud);
            PlayerPrefs.SetFloat("Long", longuitud);
            PlayerPrefs.SetFloat("Radius", radio);

            //Retirando los objetos del HUD
            GetComponentInChildren<TextMesh>().text = "Welcome to "+contexto+"\n"+"LA
T: "+latitud+"\n"+"LONG: "+longuitud;
            deslizador.SetActive (false);
            input.SetActive (false);
            texto2.enabled = false;
            HUD.GetComponent<Text> ().text = "Retire su vista del marcador para visua
lizar los recursos asociados";

        }
        else
        {
            print ("ingreso esta en " + ingreso);
            if(ingreso != false){
            close.GetComponentInChildren<AudioSource>().Play();
                print("se perdio un marcadooooooooor");
                SceneManager.LoadScene (6);
            }
        }
    }
}
```

Algoritmo 1. Tracking.cs

Este script se utiliza para detectar el marcador “Sala”, el radio de acción y obtener las coordenadas GPS actuales del dispositivo móvil, luego almacenarlas en la base de datos interna de Unity para que queden disponibles en las demás escenas, además de gestionar los elementos UI *user interface* del Canvas, son los elementos de interacción de la cabecera de pantalla o *HUD – Head Up Display*. Se ubica dentro del Objeto *ImageTarget* como se observa en la figura 21.

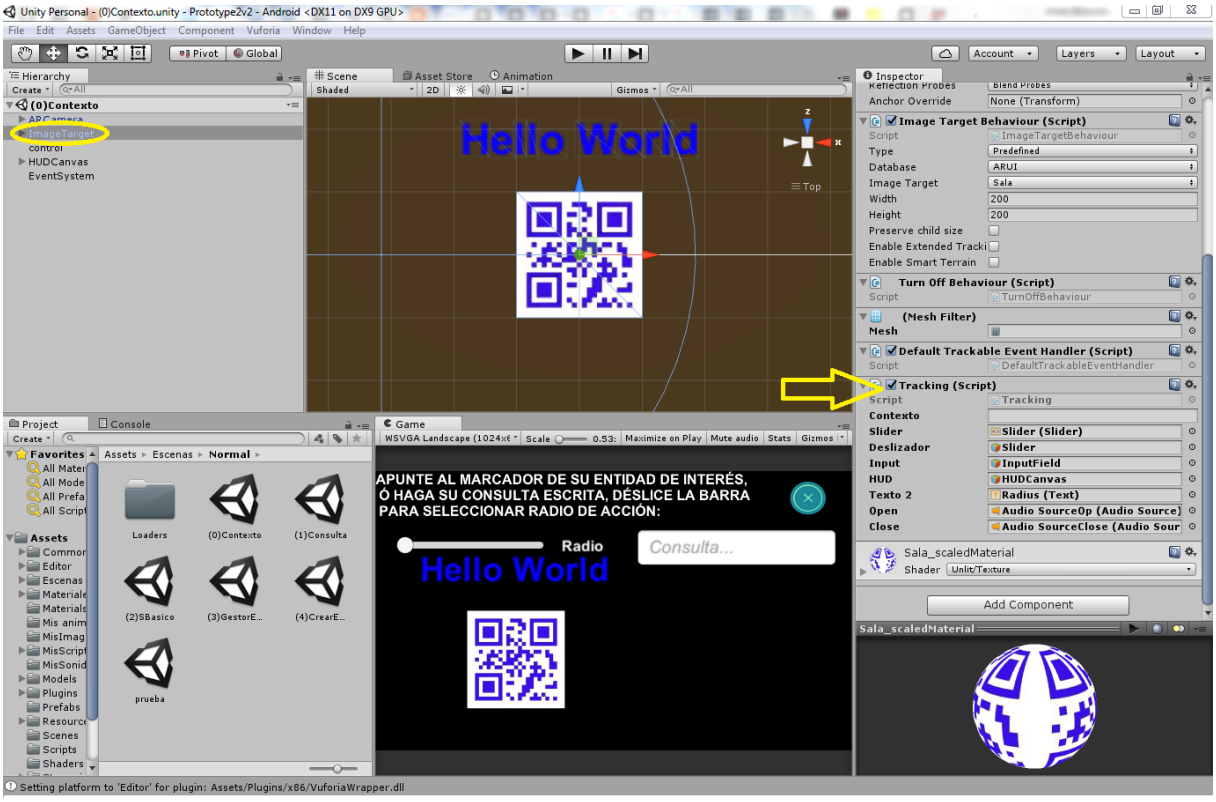

Figura 21. Ubicación del script *Tracking.cs* en Unity 3D.

Este script es la base del funcionamiento de otros scripts que tienen una estructura similar los cuales son:

- ***TargetDetector.cs*** : Script que detecta los marcadores en la escena de gestión de servicios de interacción. Habilita o Deshabilita el script ***EtiquetasECA.cs.***

CÓDIGOS RELACIONADOS CON INSTANCIACIÓN DE OBJETOS.

- ***Consulta.cs***

  Para este prototipo se utilizan los siguientes métodos del servicio web:

  ExpandirConsultaConceptosOntologia(): Este método recibe la consulta relacionada y a través de la ontología respectiva, asocia conceptos afines a la misma, ampliando la consulta y definiendo mejor el contexto del dominio de trabajo, sus argumentos son una palabra concepto y un idioma que puede ser Inglés, Español o Ambos.

  RetornarMapaLugar(): Este método realiza la búsqueda de recursos IoT con los siguientes parámetros:

  -Radio de acción en Kms.
  -Coordenadas GPS (con la ubicación de la oficina 422 en este caso: Latitud 2.446 y longitud -76.59 dentro de la Facultad de ingeniería Electrónica de la Universidad del Cauca en

Popayán) si se hace de manera definida, esto es necesario cuando se realizan las pruebas en un pc de escritorio.
-La consulta expandida creada en el método anterior.
-La palabra “Ambos” para realizar la búsqueda tanto en idioma español como en inglés.

El método **Buscar()** es muy general pero puede servir para consultas simples si se requiere utilizarlo.

Una vez obtenida la información referente a cuantos recursos IoT han sido hallados, se procede a instanciarlos o crearlos en la escena utilizando un objeto tipo prefabricado con las características y scripts necesarios, en este caso su referencia es el GameObject *prefab* , el cual puede ser referenciado a dicho objeto en el motor de desarrollo Unity 3D.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using AssemblyCSharp.DomoIndex;   //Clase asociada del servicio web:  Se encuentra en
 Reference.cs

public class Consulta : MonoBehaviour {

    public GameObject Canvascons;
    public GameObject[] promt;
    public GameObject prefab;
    public GameObject[] oi;
    public float gridX=1f;
    public float spacing = 5f;
    public string concepto,idioma;
    public double lat, lon;
    public float Rad;
    public FeedXively[] mapalugar;
    public Datastream[] datosrecurso;
    public string NumObjs;
    public Slider slider;

    void OnEnable () {

        //proceso de extracción los datos enviados desde la escena anterior
        concepto = PlayerPrefs.GetString ("Context");
        lat = PlayerPrefs.GetFloat ("Lat");
        lon = PlayerPrefs.GetFloat ("Long");
        Rad= PlayerPrefs.GetFloat ("Radius");

        //creacion del objeto del servicio web
        WSSemanticSearch ws = new WSSemanticSearch ();

        //Proceso de expandir la consulta
        string consultaexpandida = ws.ExpandirConsultaConceptosOntologia(concepto, id
ioma);

        //Proceso busqueda de recursos IoT por latitud y longuitud en un radio defini
do en Kms
        //mapalugar=ws.RetornarMapaLugar (2.446,-
76.59,consultaexpandida,idioma,Rad);
        mapalugar=ws.RetornarMapaLugar (lat,lon,consultaexpandida,idioma,Rad);
        NumObjs=mapalugar.Length.ToString();

        //imprime numero de recursos IoT encontrados en la ubicacion outdoor definida
```

```
        Canvascons=GameObject.Find ("CanvasHUD");
        Canvascons.GetComponent<Text> ().text ="Lat:"+lat+" "+"Long:"+lon+"\n"+ NumOb
js+" recursos en radio de accion "+Rad+" Km.";
        promt = new GameObject[mapalugar.Length];

        //Instancias de los objetos representacion de los recursos IoT encontrados
        for (int x = 0; x < mapalugar.Length; x++) {
            datosrecurso = mapalugar [x].feed.datastreams;
            Vector3 pos = new Vector3(x,0, 1.2f) * spacing;
            promt[x]=Instantiate(prefab,pos,transform.rotation) as GameObject;
            promt[x].GetComponentInChildren<Metadato> ().Feed =mapalugar[x].feed.id;
            promt[x].GetComponentInChildren<Metadato> ().enabled = true;
            promt[x].GetComponent<Feed> ().IDfeed=mapalugar[x].feed.id;

            for (int y = 0; y < datosrecurso.Length; y++) {
                //datos del primer recurso encontrado
                promt[x].GetComponent<Feed> ().rec = datosrecurso[1].id;
                promt[x].GetComponent<Feed> ().unds = datosrecurso[1].unit.symbol;
                promt[x].GetComponent<Feed> ().enabled = true;
             }
        }
    }

    void OnDisable(){
        //Destruccion de los objetos
        oi = GameObject.FindGameObjectsWithTag("RecursoIoT");
        for (int y = 0; y < oi.Length; y++) {
            Destroy (oi[y]);
        }

    }
}
```

Algoritmo 2.Consulta.cs

- ***EtiquetasECA.cs***

Este script es el encargado de construir los objetos referentes a las etiquetas que representan los servicios de interacción o ECA's encontrados en el directorio ECA a través del script ***ConsultaDirECA.cs***.

Los objetos que se crean en la escena a partir de un prefabricado que contiene la etiqueta con el nombre y la opción de borrar el servicio de interacción, y todas las funcionalidades que permiten esta gestión, el GameObject *prefab* se utiliza para referenciar el prefabricado en el motor de desarrollo Unity 3D.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class EtiquetasECA : MonoBehaviour {
    public int NumEcas;
    public string[] direkas=null;
    public GameObject[] quad;
    public GameObject prefab;
    public GameObject canvashud;
    public float space;
    public string msgdir;
    public string Feedobj=null;
```

```
    public int x=0;
    public GameObject Label;

    void OnEnable(){

       GetComponent<ConsultaDirECA> ().enabled = false;

        if (direkas[0] !="") {
            for (x = 0; x < NumEcas; x++) {
                Vector3 pos = new Vector3 (-300, x * space, x*space);
                Label = Instantiate (prefab, pos, transform.rotation) as GameObject;
                Label.name = "eti" + x;
                Label.GetComponentInChildren<TextMesh> ().text = direkas [x];
                Label.GetComponentInChildren<SBEca> ().id_objeto = Feedobj;
                Label.GetComponentInChildren<SetEventoState> ().NombreECA = direkas [
x];
                Label.GetComponentInChildren<SetAccionState> ().NombreECA = direkas [
x];
                Label.GetComponentInChildren<BorrarEvento> ().NombreECA = direkas [x]
;
                Label.GetComponentInChildren<BorrarAccion> ().NombreECA = direkas [x]
;
                GetComponent<CorrutinaLabel> ().enabled = true;
            }
        } else {
            canvashud.GetComponent<Text> ().text ="NO HAY ECAS ASOCIADOS AL OI";
        }
    }

    void OnDisable(){
        GetComponent<ConsultaDirECA> ().enabled = true;
        canvashud.GetComponent<Text> ().text = "SI ACTIVA UN SERVICIO DE INTERACCIÓN,
"+"\n"+"SE APAGARÁ EL SERVICIO BÁSICO DEL OBJETO.";
        GetComponent<CorrutinaLabel> ().enabled = false;
        quad = GameObject.FindGameObjectsWithTag("quad");
        for (int y = 0; y < quad.Length; y++) {
        Destroy (quad[y]);
        }
    }
}
```

Algoritmo 3.EtiquetasECA.cs

CÓDIGOS DE LAS SUSCRIPCIONES Y PUBLICACIONES A TRAVÉS DEL BRÓKER MQTT.

A continuación se presenta un código genérico que representa a todos los códigos utilizados en este desarrollo para realizar las peticiones MQTT y la recepción de los mensajes asociados a dichas peticiones, ya que su funcionalidad y estructura es la misma bajo algunas excepciones no relevantes.

- ***Query.cs***

```
using UnityEngine;
using System.Collections;
using uPLibrary.Networking.M2Mqtt;
using System;
using System.Runtime.InteropServices;
using System.Text;
using uPLibrary.Networking.M2Mqtt.Messages;
```

```
public class Query : MonoBehaviour {

    private MqttClient4Unity client;                //http://tdoc.info/blog/2014/11/
10/mqtt_csharp.html
    public string brokerHostname = null;
    public int brokerPort = 1883;
    public string userName = null;
    public string password = null;
    private Queue msgq = new Queue();

    public string IDfeed;                           //Identificacion Feed
    public string rec = null;                       //Metadato recurso
    public string unds = null;                      //Metadato unidades
    public string sustopic =null;                   //Topico de suscripcion
    public string pubtopic =null;                   //Topico de publicacion
    public string msg;                              //Mensaje recibido
    public string query;                            //Cadena tipo XML para realizar
la peticion

    void OnEnable () {
        sustopic = IDfeed + "/canal_suscripcion";     //Aqui se construye la cadena d
el canal de suscripcion segun la respuesta
        pubtopic = IDfeed + "/canal_publicacion";     //Aqui se construye la cadena d
el canal de publicacion segun la peticion

        if (brokerHostname != null && userName != null && password != null) {

            Connect ();                             //Conexion MQTT
            client.Subscribe(sustopic);                //Suscripcion
        }
            client.Publish(pubtopic, System.Text.Encoding.ASCII.GetBytes("Dummy 123..
")); //Cadena Dummy
            client.Publish(pubtopic, System.Text.Encoding.ASCII.GetBytes(query));
      //Publicacion
    }

    void Update () {
        if (client.Count() > 0) {                      //Ciclo para la recepcion de
mensajes MQTT
            string s = client.Receive();
            msgq.Enqueue(s);
            Debug.Log("received :" + s);
            msg = s;
        }
    }

    public void Connect()                           //Metodo para realizar la Conex
ion y la suscripcion
    {
        // SSL使用時はtrue、CAを指定
        client = new MqttClient4Unity(brokerHostname, brokerPort, false,null);
        // clientidを生成
        string clientId = Guid.NewGuid().ToString();
        client.Connect(clientId, userName, password);
    }

    public void Publish(string _topic, string msg)    //Metodo para publicar
    {
        client.Publish(_topic, Encoding.UTF8.GetBytes(msg),MqttMsgBase.QOS_LEVEL_AT_M
OST_ONCE, false); // retainがfalse
```

```
        }
    }
```

Algoritmo 4. Query.cs

Los algoritmos basados en este script son:

- ***Feed.cs:*** Se utiliza para suscribirse a los datos publicados por el OI en todo momento.

- ***Metadato.cs:*** Algoritmo para obtener los metadatos del OI, a través de la petición MetaDataQuery.xml, también recibe la respuesta *MetaData.xml* y le realiza un procesamiento inicial para obtener los metadatos a través del parser XML referenciado.

- ***ServicioBasico.cs:*** Su función es cambiar el estado del servicio básico del objeto inteligente y recibir la confirmación del cambio de estado a través de un mensaje tipo cadena "200" para confirmación o "404" cuando no se cambió el estado mediante el mensaje ***SimpleResponse.xml.***

- ***Metacons.cs:*** igual a *Metadato.cs* con la lógica interna relacionada con el estado del servicio básico del objeto asociado, recibe la respuesta ***MetaData.xml*** y extrae le estado del servicio básico.

- ***ConsultaDirECA.cs:*** Publica la palabra petición "**ecadir_ok**" para obtener como respuesta una cadena con los nombres de los servicios de interacción ECA's disponibles en el directorio ECA del OI. Este script fracciona la cadena cada vez que encuentra una coma, y guarda las cadenas encontradas en un arreglo de cadenas.

- ***ConsultaStateECA.cs:*** Este código publica la petición con el nombre del servicio de interacción "**nombre_eca.xml**", obteniendo como respuesta la cadena con el estado del servicio ECA en el prototipo1, y el archivo xml del respectivo Servicio ECA en el prototipo2.

- ***SetStateEca.cs, SetEventoEca.cs y SetAccionEca.cs*** son los scripts utilizados para cambiar el estado del servicio de interacción ECA en los prototipos 1 y 2, publicando la petición ***SetECAState.xml.***

- ***BorrarEca.cs, BorrarEvento.cs y BorrarAccion.cs*** son los scripts que se utilizan para borrar los servicios de interacción ECA en los prototipos 1 y 2, publicando la petición ***ECADelete.xml***.

- ***EcaEvento.cs y EcaAccion.cs*** se utilizan para crear los servicios de interacción ECA publicando el archivo **ECA.xml.**

- **RebootE.cs y RebootA.cs** para reiniciar el OI publicando la cadena "reboot_ok"

En algunos casos como la consulta de metadatos se debe tener en cuenta que el escenario desecha dos peticiones consecutivas idénticas, por eso si la consulta de los metadatos es constante, se debe publicar una cadena cualquiera tipo *Dummy* en el coordinador entre cada petición.

CÓDIGOS DE LAS INTERFACES UI DE UNITY PARA EL HUD.

- **Zoom.cs**

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;              //Libreria para poder utilizar los elementos UI de
Unity

public class Zoom : MonoBehaviour {
    GameObject Canvas;             //HUD implementado
    GameObject Botonurl;           //Boton para el vinculo web que redirige al usuario
 a los metadatos de Xively

    void OnMouseDown(){              //Captura del evento Touch o Click

        string msgmeta = GetComponentInChildren<Metadato> ().metxml;  //mensaje desde
 la peticion metadato
        GetComponent<AudioSource>().Play();

        if (msgmeta != "") {      //Captura de la excepcion de mensaje vacio

            Botonurl=GameObject.Find ("ButtonURL");
            Botonurl.GetComponent<Image> ().enabled = true;
            //Cadena que arma el vinculo a los metadatos Xively
            Botonurl.GetComponentInChildren<Text> ().text ="https://www.personal.xive
ly.com/feeds/"+GetComponentInChildren<Metadato> ().Feed;
            Botonurl.GetComponentInChildren<Text> ().enabled = true;
            GetComponent<Renderer> ().material.color = Color.green;
            Canvas = GameObject.Find ("CanvasHUD");
            Canvas.GetComponent<Text> ().color = Color.green;
            Canvas.GetComponent<Text> ().text = "Feed: "+GetComponentInChildren<Metad
ato> ().id + " " + GetComponentInChildren<Metadato> ().nombre+"\n"+"Seleccione este l
ink para ver todos sus metadatos: ";
            StartCoroutine (Deselected ());
        } else {
            Botonurl = GameObject.Find ("ButtonURL");
            Botonurl.GetComponent<Image> ().enabled = true;
            Botonurl.GetComponentInChildren<Text> ().text ="https://www.personal.xive
ly.com/feed/"+GetComponentInChildren<Metadato> ().Feed;
            Botonurl.GetComponentInChildren<Text> ().enabled = true;
            GetComponent<Renderer> ().material.color = Color.red;
            Canvas = GameObject.Find ("CanvasHUD");
            Canvas.GetComponent<Text> ().color = Color.red;
            Canvas.GetComponent<Text> ().text ="El Recurso:"+GetComponentInChildren<M
etadato> ().Feed + " " + "No es un objeto del contexto"+"\n"+"Seleccione este link pa
ra ver todos sus metadatos: ";
            StartCoroutine (Deselected ());
        }
    }
    IEnumerator Deselected() {
        yield return new WaitForSeconds(1);
        GetComponent<Renderer> ().material.color = Color.white;
    }
}
```

Algoritmo 5.Zoom.cs

ANEXO E. IMPLEMENTACION DE LAS LIBRERIAS MQTT EN UNITY.

MQTT - *Message Queue Telemetry Transport* es un protocolo abierto que sirve para dar conectividad (M2M)/"Internet of Things" con un estilo *Publish-Suscribe*, en redes de alta latencia o con restricciones.

El Broker MQTT se implementa abriendo el puerto 1883 por defecto, mediante la instalación del programa Mosquitto[1], el cual es un bróker de mensajes de código abierto que implementa MQTT en la versión 3.1.

La conexión mediante MQTT, se realizó probando algunas librerías para .Net que pudieran utilizarse en los script de C# que utiliza Unity3D; las librerías de codeplex[2], brindan un desarrollo especializado en Unity, simplemente añadiendo la librería *M2Mqtt.dll* a la carpeta de *Plugins* del proyecto se puede implementar la creación del cliente y el publicador de MQTT. El código es una combinación de los trabajos de Masatoshi Itoh, y Paolo Patierno [3].

Para comprobar que el canal se encuentra abierto se ejecuta en la consola del sistema el comando *netstat –an* verificando que el puerto 1383 o el que se seleccione para la conexión MQTT se encuentre abierto.

Se configura el script en el inspector del objeto al que se encuentre asociado, llenando las variables correspondientes como se observa en la figura 22. Para este caso se tiene como ejemplo una simple conexión al servidor de Xively siguiendo las instrucciones de su respectiva API[4] , retornando un archivo JSON con los datos del respectivo recurso o Feed.

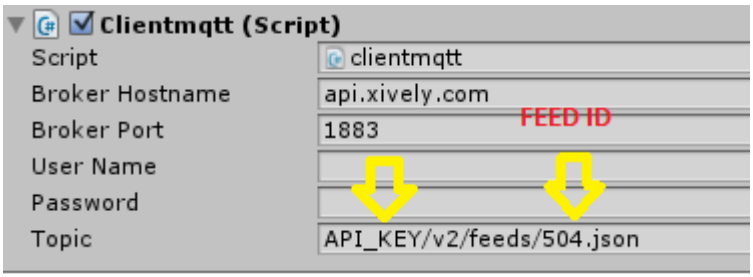

Figura 22. Configuración del Script para la conexión MQTT.

Observe que la conexión ha sido exitosa y se ha obtenido la cadena JSON deseada con los datos y metadatos del sensor del canal asociado como se observa en la figura 23, solo resta hacer un *parser* a dichos datos, cabe mencionar que no solo se pueden recibir cadenas tipo JSON sino algunos otros tipos de formato como XML entre otros.

[1] http://mosquitto.org/
[2] https://m2mqtt4unity.codeplex.com/releases/view/167009
[3] https://gist.github.com/masatoshiitoh/8b5c14d0ca2a20e3a7a1
[4] https://personal.xively.com/dev/docs/api/communicating/mqtts/

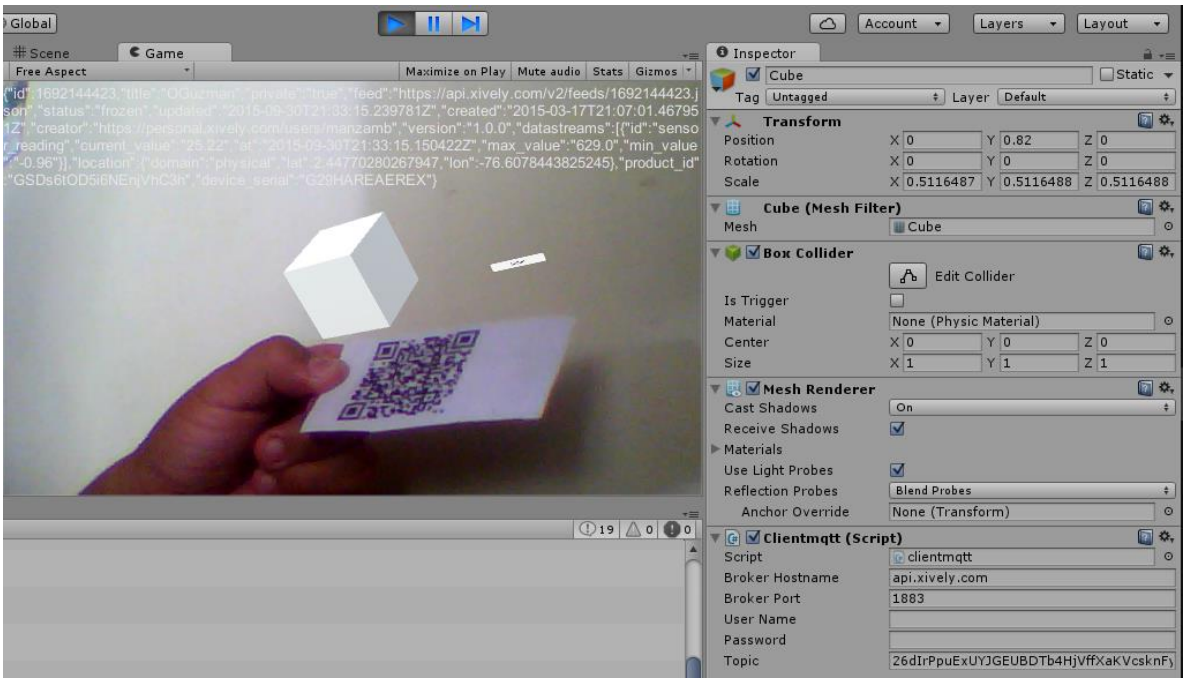

Figura 23. Conexión MQTT Exitosa.

ANEXO F. SOLICITUDES Y RESPUESTAS XML.

A continuación se detallan los mensajes utilizados en la interacción entre los recursos IoT del escenario de interacción semántica propuesto por Guerrero Riobamba [4] donde las gráficas presentadas representan una generalidad del archivo, los elementos en blanco son variables que contienen la información a comunicar, en contraste los otros elementos son estáticos.

- ***SimpleValue.xml***, entrega el valor instantáneo de una propiedad de interés captado por un recurso perteneciente a un objeto. Esta respuesta la publica el OI de manera constante en el canal: ***Feed_Objeto/Nombre_Recurso***

```
<?xml version="1.0" encoding="UTF-8"?>
<Objects>
    <Object>
        <id>id_objeto</id>
        <InfoItem name="id_recurso">
            <value type="string">valor</value>
        </InfoItem>
    </Object>
</Objects>
```

Figura 24. SimpleValue.xml. Fuente: Anexo C Escenario de interacción semántica [4].

- ***SimpleResponse.xml***, entrega un código de respuesta (200 o 404) después que el objeto termina una operación interna. Tiene el fin de informar sobre el estado de una petición realizada por el usuario. Esta respuesta la publica el OI en los canales:

    - ***Feed_Objeto /EstadoBasic,*** para la respuesta del cambio de estado del servicio básico.
    - ***Feed_Objeto/EstadoECA,*** para la respuesta del cambio de estado de servicio de interacción ECA.
    - ***Feed_Objeto/EliminarECA,*** para la respuesta al borrar el servicio de interacción ECA asociado.
    - ***Feed_Objeto/CrearECA,*** para la respuesta al crear un servicio de interacción ECA.

```
<?xml version="1.0" encoding="UTF-8"?>
<omiEnvelope>
    <response>
        <result>
            <return returnCode=code></return>
        </result>
    </response>
</omiEnvelope>
```

Figura 25 SimpleResponse.xml. Fuente: Anexo C Escenario de interacción semántica [4].

- ***SetECAState.xml***, soporta la solicitud que realiza el usuario a un objeto para el cambio del estado (prendido o apagado) de un servicio de interacción. Esta solicitud o petición se publica el OI en el canal coordinador: ***Feed_Objeto/coordinador***

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Objects>
    <Object>
        <id>id_objeto</id>
        <InfoItem name="set_eca_state">
            <InfoItem name="eca_name">
                <value>eca_name</value>
            </InfoItem>
            <value>state</value>
        </InfoItem>
    </Object>
</Objects>
```

Figura 26SetECAState.xml. Fuente: Anexo C Escenario de interacción semántica[4].

- ***SetBasicState.xml***, soporta la solicitud que realiza el usuario a un objeto para el cambio del estado (prendido o apagado) de un servicio básico. Se debe publicar en el canal coordinador así: ***Feed_Objeto/coordinador***

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Objects>
    <Object>
        <id>id_objeto</id>
        <InfoItem name="set_basic_state">
            <value>state</value>        --
        </InfoItem>
    </Object>
</Objects>
```

Figura 27SetBasicState.xml. Fuente: Anexo C Escenario de interacción semántica [4]

- ***MetaDataQuery.xml***, soporta la consulta realizada por el usuario sobre los metadatos de un objeto. El usuario debe publicar esta petición en el canal coordinador de la siguiente manera: ***Feed_Objeto/coordinador.***

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Objects>
    <Object>
        <id>id_objeto</id>
        <InfoItem name="metadata_query"></InfoItem>
    </Object>
</Objects>
```

Figura 28. MetaDataQuery.xml. Fuente: Anexo C Escenario de interacción semántica [4].

- ***ECADelete.xml***, Soporta la solicitud que realiza el usuario a un objeto para eliminar un servicio de interacción. Esta solicitud se publica en el canal coordinador así: ***Feed_Objeto/coordinador***

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Objects>
    <Object>
        <id>id_objeto</id>
        eca
        <InfoItem name="eca_delete">
            <value>nombre_eca</value>
        </InfoItem>
    </Object>
</Objects>
```

Figura 29. ECADelete.xml. Fuente: Anexo C Escenario de interacción semántica[4].

- ***ECA.xml***, Contiene toda la estructura sobre el servicio de interacción. Utilizado para crear la sentencia ECA en cada objeto participante en el servicio de interacción. Este archivo se publica en el canal coordinador así: ***Feed_Objeto/coordinador***

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Objects>
    <Object>
        <id>id_objeto</id>
        <InfoItem name="eca">
            <InfoItem name="name">
                <value type="string">nombre_eca</value>
            </InfoItem>
            <InfoItem name="state">
                <value type="string">estado_eca</value>
            </InfoItem>
            <InfoItem name="entidad_interes">
                <value type="string">entidad_interes</value>
            </InfoItem>
            <InfoItem name="evento">
                <InfoItem name="id_objeto">
                    <value type="string">id_objeto_eveto</value>
                </InfoItem>
                <InfoItem name="nombre_objeto">
                    <value type="string">nombre_objeto_eveto</value>
                </InfoItem>
                <InfoItem name="id_recurso">
                    <value type="string">id_recurso_evento</value>
                </InfoItem>
                <InfoItem name="nombre_recurso">
                    <value type="string">nombre_recurso_evento</value>
                </InfoItem>
            </InfoItem>
            <InfoItem name="condicion">
                <InfoItem name="comparador">
                    <value type="string">comparador_condicion</value>
                </InfoItem>
                <InfoItem name="variable">
                    <value type="tipo_variable_condicion">variable_condicion</value>
                </InfoItem>
                <InfoItem name="unidad">
                    <value type="string">unidad_condicion</value>
                </InfoItem>
                <InfoItem name="meaning">
                    <value type="string">significado_condicion</value>
                </InfoItem>
            </InfoItem>
            <InfoItem name="accion">
                <InfoItem name="id_objeto">
                    <value type="string">id_objeto_accion</value>
                </InfoItem>
                <InfoItem name="nombre_objeto">
                    <value type="string">nombre_objeto_accion</value>
                </InfoItem>
                <InfoItem name="id_recurso">
                    <value type="string">id_recurso_accion</value>
                </InfoItem>
                <InfoItem name="nombre_recurso">
                    <value type="string">nombre_recurso_accion</value>
                </InfoItem>
                <InfoItem name="comparador">
                    <value type="string">comparador_accion</value>
                </InfoItem>
                <InfoItem name="variable">
                    <value type="tipe_variable_accion">variable_accion</value>
                </InfoItem>
                <InfoItem name="unidad">
                    <value type="string">unidad_accion</value>
                </InfoItem>
                <InfoItem name="meaning">
                    <value type="string">significado_accion</value>
                </InfoItem>
            </InfoItem>
        </InfoItem>
    </Object>
```

Figura 30. ECA.xml. Fuente: Anexo C Escenario de interacción semántica [4].

- ***MetaData.xml***, contiene toda la información sobre los metadatos de un objeto. Utilizado para intercambiar la información básica de un objeto con otro objeto o con el usuario. El OI publica esta respuesta en el canal: ***Feed_Objeto/MetaData***

ANEXO G. ACCESO AL DIRECTORIO DEL OBJETO INTELIGENTE EN LAS PLACAS GALILEO.

El acceso al sistema operativo Linux Yocto[5] embebido en las placas Galileo se realiza a través de la red de área local - LAN donde se encuentra el OI, esta red le asigna una dirección IP local a través de su dispositivo enrutador o *router.* Para conocer esta dirección se utilizan líneas de comando de Windows como *ipconfig* o *netstat*, las cuales brindan información poco intuitiva acerca de los miembros de la red local, tanbien se puede acceder al enrutador y observar las conexiones DHCP a través de la página web que lo gestiona, si el administrador lo permite.

Por la anterior razón se opta por utilizar una aplicación para el S.O. Android, llamada Fing[6] la cual lista de manera organizada e intuitiva los miembros de la red local y sus características además de las direcciones físicas o MAC's conectadas al enrutador local como se observa en la figura 31 con lo cual se pueden distinguir fácilmente los OI asociados a la LAN en cuestión.

Figura 31. Objetos inteligentes conectados a la red local a través de la aplicación Fing.

[5] https://www.yoctoproject.org/
[6] https://play.google.com/store/apps/details?id=com.overlook.android.fing&hl=es_419

Una vez obtenida la dirección IP del objeto de interés, se puede utilizar un emulador de terminal como Putty[7] con el cual se puede acceder a la línea de comandos de Yocto Linux como usuario: *root* como se observa en la figura 32.

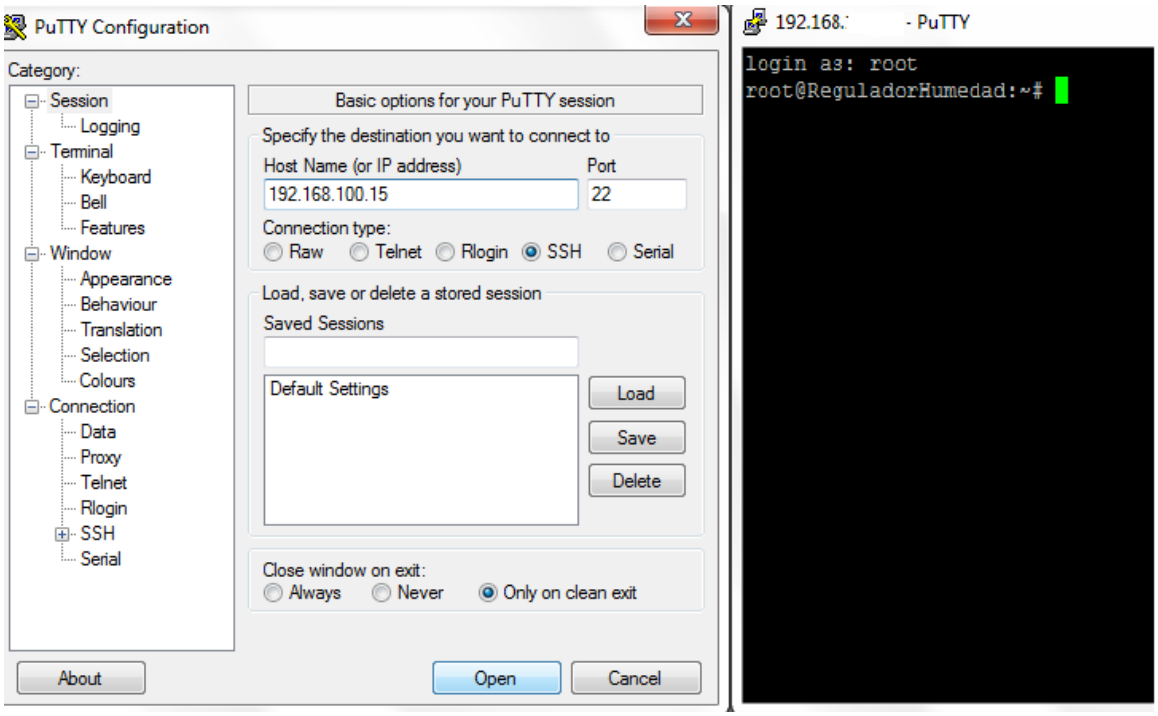

Figura 32. Emulador Terminal Putty.

Al ser Python el lenguaje de desarrollo implementado se hace necesario instalarlo en la maquina donde se va a gestionar la imagen de los OI, para este caso la versión 2.7.3, para Windows, este lenguaje se puede descargar en la siguiente página web:

https://www.python.org/downloads/

A continuación se tiene una lista de algunos de los comandos utilizados para la gestión de archivos a través de la línea de comandos de Putty, además de otros comandos que utiliza el lenguaje Python.

- **ls:** permite visualizar la lista de archivos en el directorio root.
- **ps:** permite visualizar todos los procesos activos.
- **python nombre_proceso.py:** corre el proceso en cuestión si existe el compilador de Python.
- **python -O nombre_proceso.py:** genera un directorio asociado al proceso en cuestión para ser accedido igual que una librería.
- **kill numero_proceso:** apaga el proceso asociado al número de proceso en cuestión.
- reboot: reinicia el OI.

Si se desea acceder al directorio root del OI para crear, copiar o borrar archivos del mismo, la forma más eficiente de hacerlo es a través de un cliente gestor de archivos como WinSCP[8], al igual

[7] http://www.putty.org/

que Putty se accede al directorio utilizando como usuario: *root*. Se debe seleccionar el protocolo adecuado que puede ser SSH, SFTP, SCP, FTP, para este caso será SCP como se observa en la figura 33.

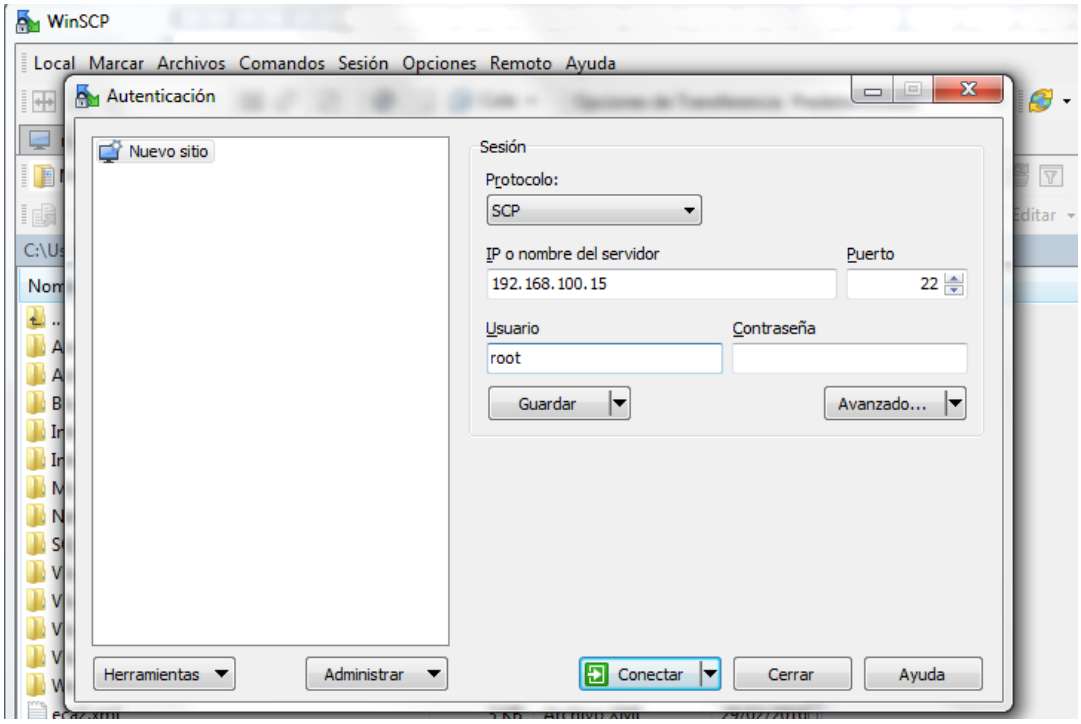

Figura 33. Cliente WinSCP.

Una vez dentro del directorio se pueden explorar y gestionar los contenidos, agregar nuevos archivos y /o editar los existentes como se observa en la figura 34.

[8] https://winscp.net/eng/download.php

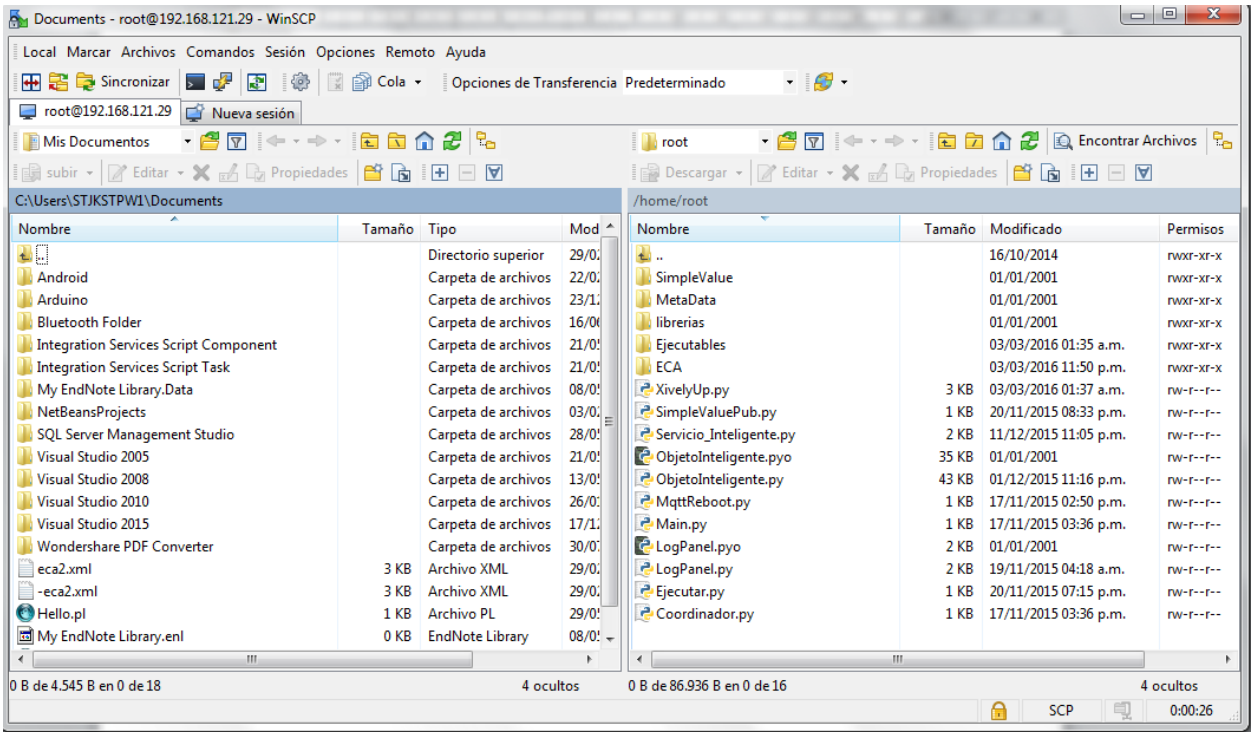

Figura 34. Archivos en el directorio Root del OI.

ANEXO H. PROGRAMAS EN PYTHON.

- ***Ecadirinfo.py***

  Este código permite obtener la lista de archivos alojados en el directorio */home/root/ECA* del OI, a través de la frase petición "**ecadir_ok**".

  El nuevo proceso define una función **acceso()** la cual que hace uso del método **listdir()** para acceder a los archivos del directorio, este método retorna una lista, que luego se convierte en cadena para ser publicada de manera adecuada o en algún formato en especial como se puede ver en la figura 35.

Ecadirinfo.py

```
1  from ObjetoInteligente import Objeto
2  import paho.mqtt.client as mqtt
3  import paho.mqtt.publish as publish
4  import os
5
6  OI = Objeto()
7  MqttClient = mqtt.Client()
8
9  id_objeto = OI.GetDiccionarioObjeto()["id_objeto"]
10 def acceso():
11     lista=os.listdir("/home/root/ECA")
12     cadena=",".join(lista)
13     publish.single(id_objeto + "/datoecadir",cadena, hostname="iot.eclipse.org")
14
15 def on_message(client, userdata, msg):
16     if(msg.payload == "ecadir_ok"):
17
18         print "prueba directorio ECA"
19             acceso()
20
21 run = True
22 while (run == True):
23     try:
24         MqttClient.connect("iot.eclipse.org", 1883, 60)
25         run = False
26     except:
27         pass
28
29 MqttClient.subscribe(id_objeto + "/ecadir")
30 MqttClient.on_message = on_message
31 MqttClient.loop_forever()
```

Figura 35. Programa Ecadirinfo.py

Este programa se implementa a través de un hilo en el directorio o librería del OI llamada ***ObjetoInteligente.py,*** este código se modifica añadiendo las siguientes líneas que definen la función del hilo como se observa en la figura 36:

```
235
236     #Se definen las funciones que trabajaran los hilos
237
238     def __Stateeca(self):
239         os.system("python " + path + "Stateeca.py")
240     def __Ecadirinfo(self):
241         os.system("python " + path + "Ecadirinfo.py")
242     def __MqttReboot(self):
243         os.system("python " + path + "MqttReboot.py")
244
245     def __ServicioInteligente(self):
246         os.system("python " + path + "Servicio_Inteligente.py")
```

Figura 36. Definición de la función del hilo en *ObjetoInteligente.py.*

Y creando la ejecución del hilo añadiendo las siguientes líneas de código a la misma librería como se observa en la figura 37.

```
528     #Ejecuta los hilos
529     def IniciarObjeto(self):
530         global nombre_objeto
531         global id_objeto
532         global Log
533
534         print "Iniciando " + nombre_objeto
535
536         reboot = threading.Thread(target=self.__MqttReboot, name='MqttReboot')
537         ecastate = threading.Thread(target=self.__Stateeca, name='Stateeca')
538         ecadir = threading.Thread(target=self.__Ecadirinfo, name='Ecadirinfo')
539         xivelyup = threading.Thread(target=self.__XivelyUp, name='XivelyUp')
540         sinteligente = threading.Thread(target=self.__ServicioInteligente, name='Se
541         simplevalue = threading.Thread(target=self.__SimpleValue, name='SimpleValue
542         coordinador = threading.Thread(target=self.__Coordinador, name='Coordinador
543         ejecutar = threading.Thread(target=self.__Ejecutar, name='Ejecutar')
544
545         try:
546             ecastate.start()
547             Log.PubRawLog(id_objeto, id_objeto, "Iniciando Stateeca")
548         except:
549             Log.PubRawLog(id_objeto, id_objeto, "Problema al Iniciar Stateeca")
550         try:
551             ecadir.start()
552             Log.PubRawLog(id_objeto, id_objeto, "Iniciando Ecadirinfo")
553         except:
554             Log.PubRawLog(id_objeto, id_objeto, "Problema al Iniciar Ecadirinfo")
555
```

Figura 37. Ejecución del hilo en *ObjetoInteligente.py.*

Finalmente utilizando Putty, se ingresa al OI, se copia el script ***Ecadirinfo.py*** en su directorio raíz y se crea nuevamente la referencia al directorio ***ObjetoInteligente.py,*** si se desea se puede borrar la referencia anterior llamada ***ObjetoInteligente.pyo.***

Para crear la referencia, se verifica que en la lista del directorio se encuentren los archivos ***Ecadirinfo.py*** y ***ObjetoInteligente.py*** mediante el comando ***ls,*** luego se ingresa el siguiente comando: *python –O ObjetoInteligente.py* , si no hubieron errores en la creación de la referencia como se observa en la figura 38, se procede a reiniciar el OI a través del comando *reboot*, se puede verificar que se ha creado una nueva referencia ***ObjetoInteligente.pyo.***

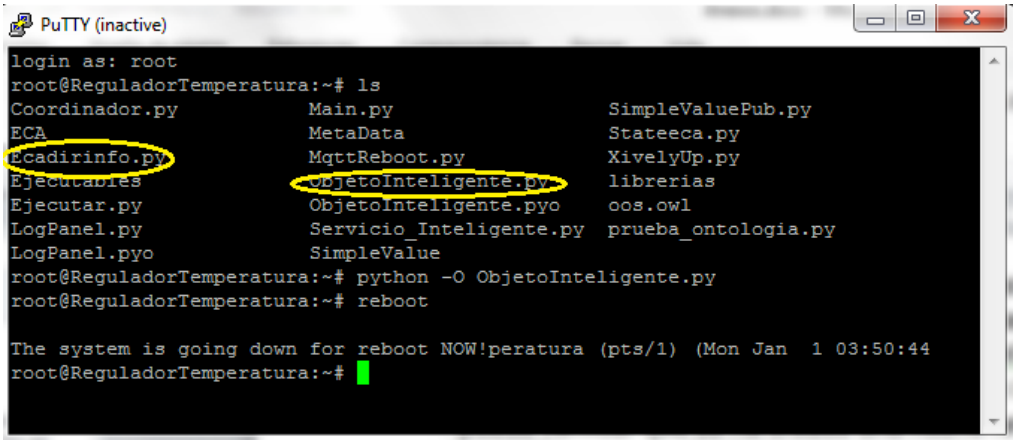

Figura 38. Creación de la referencia *ObjetoInteligente.pyo.*

La dinámica de la comunicación a través del bróker MQTT se establece al publicar la cadena "***ecadir_ok***" en el canal:

Feed_Objeto/ecadir

El OI responde con una cadena separada con comas, con los nombres de los archivos encontrados en el directorio ECA a través del canal:

Feed_Objeto/datoecadir

- ***Stateeca.py***

Este código obtiene el estado de cada uno de los archivos xml alojados en el directorio ECA del OI, a través de la petición creada con el nombre del archivo xml asi: "**nombre _eca.xml**".

Este proceso utiliza la funcion ***Acceso(nombre_eca)*** con el nombre del archivo ECA como argumento, dentro de esta se hace un parseo de dicho xml en la ruta predefinida, y se recorre buscando coincidencia con el nodo "*InfoItem*", para encontrar el estado del ECA, y luego publicarlo como se observa en la figura 39.

Su implementacion es igual a la implementacion del codigo *Ecadirinfo.py,* sobre el *ObjetoInteligente.py,* y su dinamica de comunicación a traves del broker MQTT se establece al publicar el nombre del archivo ECA "nombre_eca.xml" con su extension en el canal:

Feed_Objeto/ecastate

El OI responde con una cadena que contiene el estado "*on*" o "*off*" según sea el caso, en el canal:
Feed_Objeto/datoecadir

Stateeca.py

```
1  from ObjetoInteligente import Objeto
2  import paho.mqtt.client as mqtt
3  import paho.mqtt.publish as publish
4  import os
5  from xml.dom import minidom
6  from xml.dom.minidom import parse, parseString
7  OI = Objeto()
8  MqttClient = mqtt.Client()
9
10 id_objeto = OI.GetDiccionarioObjeto()["id_objeto"]
11 def acceso(n_eca):
12     xmldoc = parse("/home/root/ECA/"+n_eca)
13     infoi = xmldoc.getElementsByTagName("InfoItem")
14     for e in infoi:
15      tag = e.getAttribute("name")
16      if (tag == "state"):
17        tagv=e.getElementsByTagName("value")[0]
18        estadoek=tagv.firstChild.data
19        print estadoek
20     publish.single(id_objeto + "/datoecaestate",estadoek, hostname="iot.eclipse.org")
21
22 def on_message(client, userdata, msg):
23     print "prueba estado ECA"
24     name_eca= msg.payload
25     acceso(name_eca)
26 run = True
27 while (run == True):
28     try:
29         MqttClient.connect("iot.eclipse.org", 1883, 60)
30         run = False
31     except:
32         pass
33
34 MqttClient.subscribe(id_objeto + "/ecastate")
35 MqttClient.on_message = on_message
36 MqttClient.loop_forever()
```

Figura 39. Programa Stateeca.py. Fuente propia.

- ***Stateeca.py versión 2.***

Ante la necesidad de obtener más información del archivo xml, se opta por crear otra versión del código ***Stateeca.py*** como se observa en la figura 40, con igual funcionamiento pero a diferencia de la primera versión, este código publica todo el archivo xml del servicio de interacción, pues es más sencillo hacer su análisis a través del parser xml, en la aplicación.

Stateeca.py

```
from ObjetoInteligente import Objeto
import paho.mqtt.client as mqtt
import paho.mqtt.publish as publish
import os
from xml.dom import minidom
from xml.dom.minidom import parse, parseString
OI = Objeto()
MqttClient = mqtt.Client()

id_objeto = OI.GetDiccionarioObjeto()["id_objeto"]
def acceso(n_eca):
    archivo = open("/home/root/ECA/"+n_eca,"r")
    ecafile=archivo.read()
    publish.single(id_objeto + "/datoecastate",ecafile, hostname="iot.eclipse.org")
    archivo.close()
def on_message(client, userdata, msg):
    print "prueba estado ECA"
    name_eca= msg.payload
    acceso(name_eca)
run = True
while (run == True):
    try:
        MqttClient.connect("iot.eclipse.org", 1883, 60)
        run = False
    except:
        pass

MqttClient.subscribe(id_objeto + "/ecastate")
MqttClient.on_message = on_message
MqttClient.loop_forever()
```

Figura 40. Programa Stateeca.py versión 2. Fuente propia.

ANEXO I. PREFABRICADOS

- **OIX.**

  Este prefabricado se utiliza para instanciar las representaciones graficas de los recursos IoT encontrados a través del script ***Consulta.cs*** en la escena 2.

  La creación de un prefab se encuentra descrita en los tutoriales de Unity en los siguientes enlaces:

  https://docs.unity3d.com/es/current/Manual/Prefabs.html

  https://unity3d.com/es/learn/tutorials/topics/interface-essentials/prefabs-concept-usage

  y su instanciación en:

  https://docs.unity3d.com/es/current/Manual/InstantiatingPrefabs.html

  OIX se compone de un objeto 3D, y dos etiquetas de texto llamadas *NombresText y RecursosText* , para presentar los datos y metadatos del primer recurso del OI como se observa en la figura 41. Estos prefabricados se almacenan en el directorio raíz de Unity.

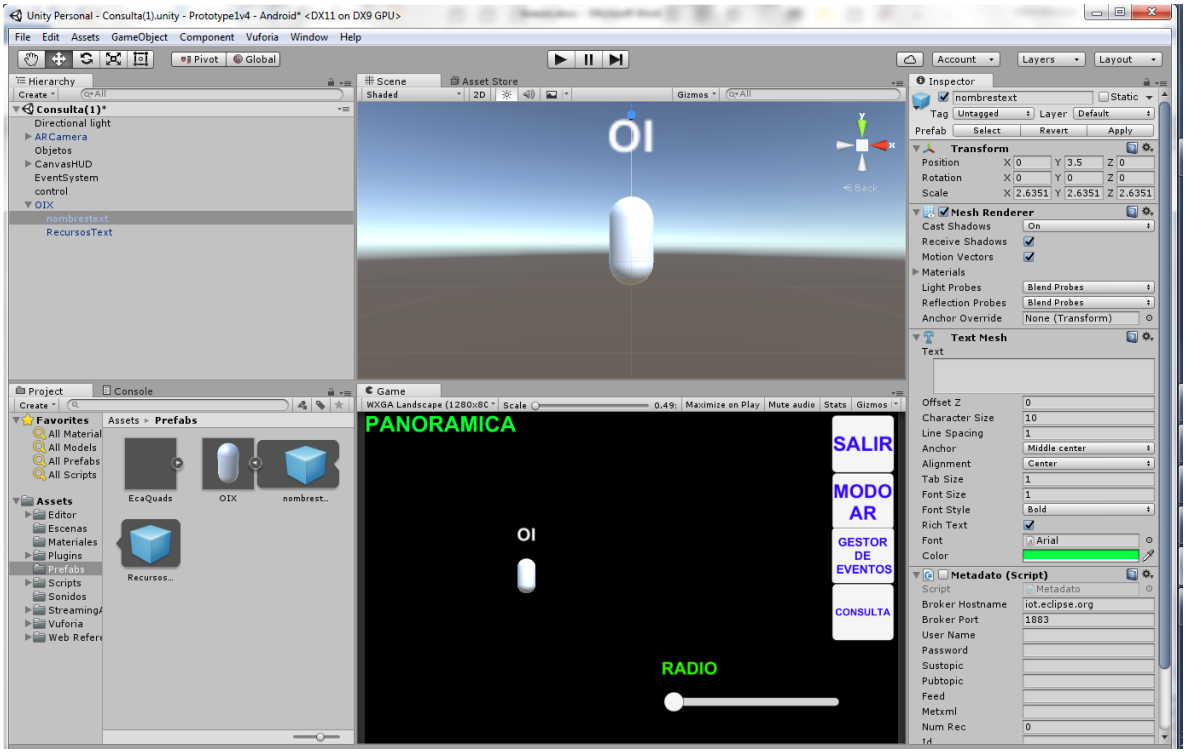

Figura 41. Estructura del prefabricado OIX.

NombresText obtiene los metadatos del recurso del OI a través del script ***Metadato.cs,*** y **RecursosText** hace uso del script ***EtiquetaSensor.cs*** para el color de los textos y el dato del OI.

El dato del objeto inteligente y el zoom semántico lo obtiene el Modelo 3D a través de los scripts ***Feed.cs*** y ***Zoom.cs***.

- ***EcaQuads***

Este prefabricado despliega las etiquetas para poder gestionar los servicios de interacción en la escena 4, se compone de 3 etiquetas llamadas ***Quad***, ***BorrarQuad*** e ***InfoQuad***, las cuales muestran el nombre del ECA con el estado, la acción de borrar el servicio de interacción, y la información sobre el mismo, respectivamente. La figura 42 muestra la etiqueta genérica que instancia el script ***EtiquetasECA.cs.***

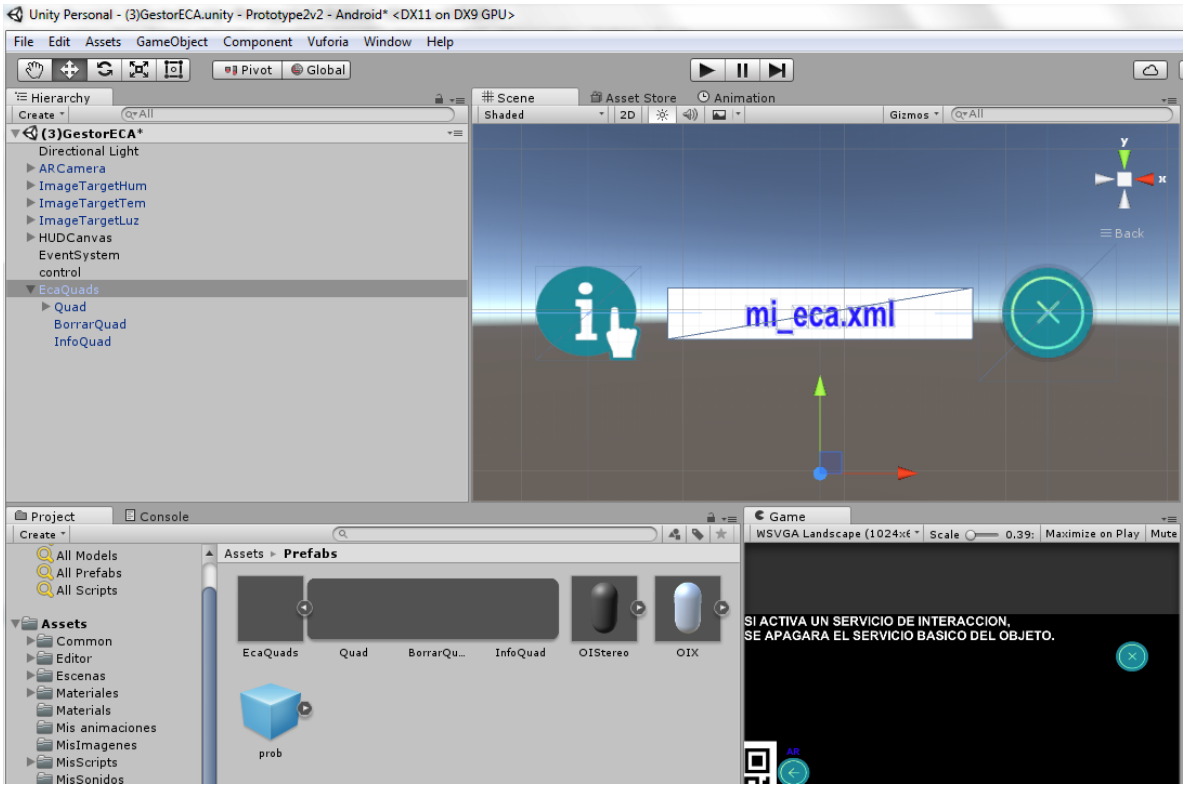

Figura 42. Prefabricado EcaQuads.

El objeto ***Quad*** es una etiqueta que despliega el nombre del servicio de interacción enviado por ***EtiquetasECA.cs,*** implementando la funcionalidad de poder cambiar el estado del mismo a través de los script ***SetEcaState.cs*** en el primer prototipo, y los script ***SetEventoState.cs*** y ***SetAccionState.cs*** para el segundo prototipo como se observa en la figura 43. La funcionalidad del cambio de estado al tocar o hacer click en la etiqueta es controlada por el script **ToqueSet.cs.**

Figura 43. Lista de Scripts del objeto Quad. Fuente Propia.

El objeto **BorrarQuad,** es la etiqueta que permite la funcionalidad de borrar el servicio de interacción a través del script ***BorrarEca.cs*** en el prototipo 1, y ***BorrarEvento.cs, Borrar Accion.cs*** en el prototipo2 como se observa en la figura 44. Estos scripts son controlados por el script ***ToqueBorrar.cs.***

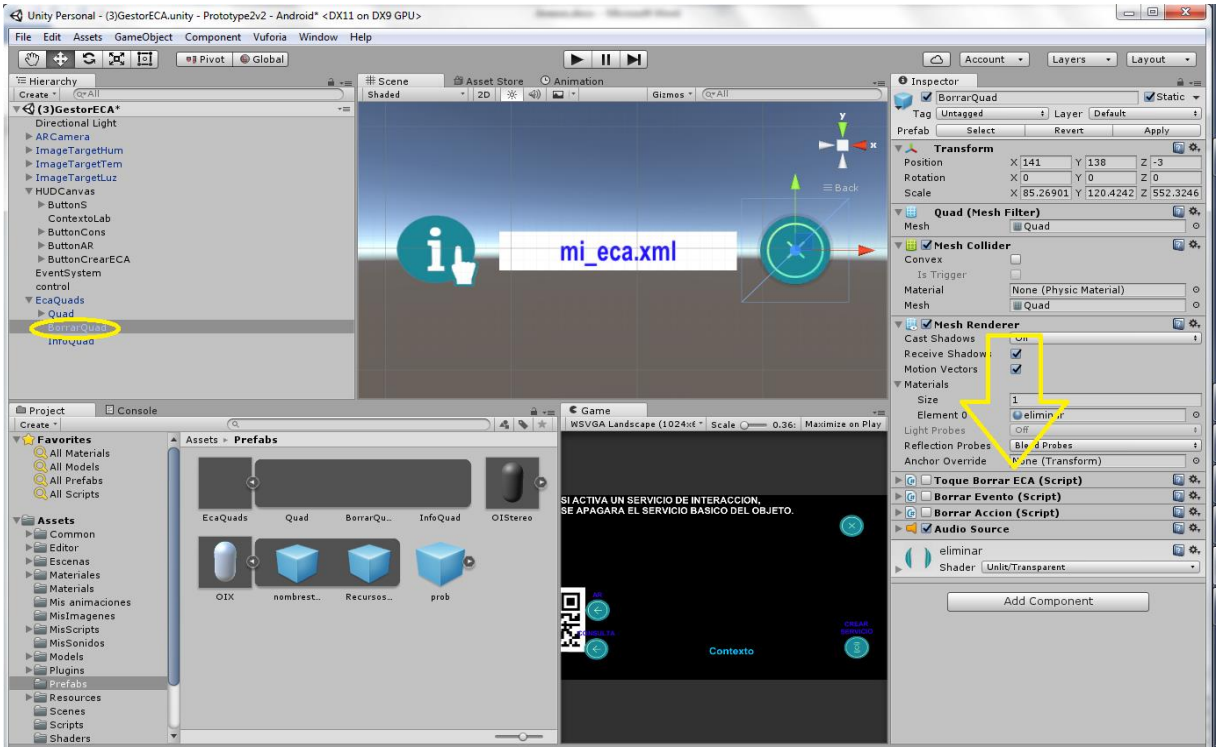

Figura 44. Lista de scripts de BorrarQuad.

Finalmente el objeto **InfoQuad,** es la etiqueta que brinda información acerca de la función del ECA respectivo, utilizando el script ***ToqueInfoECA.cs,*** el cual muestra la cadena que describe el servicio de interacción respectivo generada en ***ConsultaStateECA.cs*** y transmitida por ***CorrutinaLabel.cs.***

ANEXO J. PRUEBA DE VALIDACION DEL PROTOTIPO 1.

Pruebas por cada iteración.

En este prototipo la capacidad de la aplicación de realizar las funcionalidades básicas del escenario IoT es la prioridad, estas funcionalidades son:

- *Contextualizar o realizar una consulta al índice semántico.*
- *Consultar los datos y metadatos de un recurso IoT tanto en el índice semántico asociado, como en el OI respectivo.*
- *Cambiar el estado de un servicio básico asociado al OI.*
- *Cambiar el estado de un ECA.*
- *Borrar un ECA.*
- *Crear un ECA.*

La metodología propuesta para el desarrollo del prototipo 1 sugiere hacer las pruebas a cada escena a lo largo del proceso de desarrollo para la retroalimentación entre cliente y desarrollador y una prueba final de cada prototipo, a continuación se detalla las observaciones encontradas por el rol de cliente/usuario, asumido tanto por el director del proyecto como el desarrollador.

Se pretender realizar unas pruebas fundamentadas, informales y rápidas teniendo siempre en cuenta que la interfaz se encuentra desplegada sobre un escenario de interacción semántica, que no es un producto final, sino una prueba de concepto, por lo tanto ya de por si se tienen consideraciones y características que pueden generar conflictos con el desarrollo de la interfaz de AR para la IoT.

- **Observaciones Escena 1:**

  En esta escena se realizó con éxito la tarea de contextualizar al usuario en un escenario específico, cuando la consulta se hace a través de código QR, o cuando ingresa una palabra para la consulta.

  Se puede implementar un menú que permita escoger entre un modo monocular o uno estereoscópico, e incluso si es posible poder escoger un índice para la consulta, también se sugiere establecer una opción para realizar la consulta de manera simple o expandida.

  Las ayudas auditivas también son un aspecto enriquecedor que se puede implementar en la aplicación.

  Cuando la consulta se hace a través de código QR en modo normal, se le debe indicar al usuario que apunte la cámara del dispositivo movil fuera del código para contextualizar.

- **Observaciones Escena 2:**

  En esta escena, las representaciones de los recursos encontrados en la consulta permanecen estáticas en la pantalla, pero se podría hacer uso de los datos que brinda el índice como coordenadas GPS y distancias al objeto que consulta, los sensores internos del dispositivo móvil como el acelerómetro, el magnetómetro, el giroscopio para que las representaciones

virtuales se posicionaran sobre el campo de visión dinámicamente, al mover el dispositivo, estos sensores también pueden brindar información adicional al usuario a través de la implementación de un mini mapa o un radar ubicados en el HUD.

Se puede probar el uso de dispositivos como BLE para estimar distancias sin consultar al índice haciéndolas más precisas en interiores, e incluso poder hacer algún tipo de triangulación.

Los metadatos que entrega el índice son bastantes y pueden saturar de información la pantalla, se podría indicar al usuario un enlace web al middleware para ver los metadatos completos.

Se requiere más información al usuario sobre la naturaleza de la escena, sobretodo en el slider del radio de la consulta, pero sin saturar el HUD.

- **Observaciones Escena 3:**

  El escenario de interacción semántica carece del servicio de apagar los actuadores a voluntad, se debería implementar un hilo en el OI, que permita esta funcionalidad.

  El reflejo natural del usuario es querer tocar los objetos, en este sentido se pueden implementar botones virtuales para la vista en modo estereoscópico,

  Se debe informar al usuario de alguna manera acerca del servicio básico, en que consiste y cuáles son los umbrales donde actúa.

- **Observaciones Escena 4**:

  En esta escena se realizó con éxito las tareas de gestión de ECA's, como lo son el cambio de estado de cada ECA, y su borrado, de manera dinámica, es decir sobre el directorio interno de cada OI en el sistema operativo Linux Yocto.

  El nombre de esta escena debe ser comprensible para el usuario, se sugiere nombrarla como: Gestión de Servicios.

  Se observó que al no existir sincronía entre las peticiones de cambio de estado de los ECA's, algunas veces se generan peticiones de estado nulo, o vacio, esto perjudica notablemente la funcionalidad y se deben evitar.

  Se encontró que al instanciar los objetos tipo *Label* o Etiqueta, lo hacen fuera del *GameObject* **Imagetarget**, es decir no se instancian como hijos de dicho *GameObject* sin embargo esto no afecta la funcionalidad.

  Se debe dar información relevante del ECA a gestionar al usuario, no solo el nombre es suficiente, puede ser un zoom semántico a la hora de señalarlo.

  También se observó que ante la necesidad de apagar el servicio básico asociado del objeto Acción al momento de encender un ECA, esta función se está realizando en la siguiente escena

, lo cual es innecesario en ese punto, y sería más adecuado apagar el servicio cuando se enciende el ECA en esta escena.

Debido a que los archivos ECA se crean tanto en el directorio del objeto evento como en el del objeto acción, la funcionalidad de esta escena modifica o borra cada archivo por separado, haciendo necesario implementar la solución de manera que modifique o borre ambos archivos, cuando la acción de cambiar de estado o borrar se ejecute por parte del usuario en cualquiera de los recursos involucrados en el ECA.

- **Observaciones Escena 5.**

  En esta escena la aplicación tiene la capacidad de crear un ECA, y sus respectivos archivos tanto en el objeto evento como en el objeto acción.

  El nombre del ECA puede generarse de manera automática, pero su descripción es necesaria a la hora de gestionarlo, por eso o bien se obvia la petición de asignarle un nombre al ECA o se coloca esta petición de forma consecuente. La descripción del ECA también se hace necesaria sea de manera automática o por petición al usuario.

  Se dejó a libertad del usuario colocar los valores de la condición del objeto evento, sin embargo falta considerar que los actuadores también pueden generar un evento, por tanto es preciso ajustar la selección de tipo de valor booleano y el valor los actuadores (*on/off*) como eventos de un ECA.

  Al seleccionar el objeto acción, es necesario restringirle al usuario la posibilidad de seleccionar un sensor, por tanto se hace necesario destruir o deshabilitar este tipo de objetos en la selección del objeto acción.

  Se requiere de un botón que permita cancelar la creación del ECA en cualquier momento, y unos botones de navegación para navegar durante el proceso de creación del ECA. Los botones de selección de acción y OK deben estar ubicados consecuentemente para evitar confundir al usuario, incluso cambiar OK por la palabra CREAR ECA.

  Las etiquetas de condición y valor deben ser uniformes con el resto del prototipo.

Validación General del prototipo uno, 8 de agosto de 2016.

Como observación general para el siguiente prototipo sería adecuado añadir o modificar algunos elementos y aspectos.

Con respecto a AR el cliente sugiere asentar este concepto, fijando al objeto físico el código QR asociado haciéndolo más pequeño, o fijándole alguna textura que le brinde identidad propia de tal modo que la combinación de la realidad física y la aumentada estén ligadas sobre el objeto de interés, también se sugiere el uso de botones virtuales para mayor accesibilidad e interacción con el objeto.

La necesidad de sincronización surge a partir del hecho que la mayoría de las funcionalidades de la aplicación se basan en el protocolo de comunicación MQTT, lo cual implica un tiempo de respuesta variable, que depende de un servicio de terceros como lo es un bróker IoT, por lo tanto se debe buscar una solución en este sentido para mejorar los tiempos de respuesta de la aplicación, o por lo menos mitigarlos con una transición grafica como los iconos de espera y que el usuario no se frustre aguardando una respuesta, o durante la carga de una escena.

Es importante para el usuario saber en todo momento durante la ejecución de la aplicación sobre que contexto se encuentra, es decir en este caso el nombre de la EI, se sugiere ubicarlo en un lugar visible y consecuente con la opción de consulta al índice.

El menú del HUD puede ser simplificado por flechas de navegación, puesto que el usuario en el prototipo comprende mejor la metáfora de un navegador tipo web, en vez del uso de botones para moverse entre las escenas, además que se debe replantear la organización de estas.

Las letras y números en el HUD deben ser uniformes y con un formato adecuado, si la saturación de información es bastante, mejor no mostrarla si no es necesario, por ejemplo las cifras del GPS, y a lo largo de toda la aplicación debe estar presente la EI en alguna posición fácil de observar por el usuario, la convención de colores también debe ser uniforme, por ejemplo verde es encendido, rojo es apagado, amarillo es seleccionado.

Se debe mejorar la posición del HUD en las diferentes resoluciones de pantalla y los modos vertical u horizontal, o bloquear el modo vertical para manipular la aplicación, o si se desea poder seleccionar entre un modo estereoscópico y uno monocular.
A nivel asistencial se puede complementar el HUD con indicadores sonoros e incluso voz, animaciones, resaltando objetos y otros elementos a considerar según los recursos y el tiempo disponible.

Respecto al aspecto ergonómico se podría probar la aplicación implementado el uso de dispositivos eyewear para visión estereoscópica, pues en este primer prototipo las interacciones con la interfaz se llevan a cabo mediante el uso del touch de la pantalla y el teclado del dispositivo móvil, sosteniéndolo con una o ambas manos
,
Como ultima prioridad se puede optimizar la aplicación, apagando el GPS cuando no es requerido, indagando si el dispositivo móvil posee los sensores necesarios como son Giroscopio, GPS, acelerómetro y demás necesarios para utilizar la app, y ofrecer alternativas cuando se carece de alguno de estos.

ANEXO K. EVALUACION DE USABILIDAD DEL PROTOTIPO 2.

Se tuvo un solo grupo de usuarios cuya motivación dependió de que tan cercanos eran al área de conocimiento abarcada, y en la utilidad que le encontraron a la interfaz, en este caso al utilizar un índice de domótica en el escenario de interacción, las personas debieron estar interesadas en controlar los recursos IoT de una casa u oficina, evaluando el "como" lo hacen a través de una interfaz en AR, su nivel de satisfacción y la usabilidad de la interfaz

PROCESO DE EVALUACION DE USABILIDAD.

Fase 1. Preparación del escenario.

Las pruebas se pueden realizar individualmente en cada sujeto de prueba, sin necesidad de agruparlos en un momento en específico, esto no influirá en los resultados ya que la experiencia es individual y al contrario puede ser beneficioso para los resultados de la evaluación al eliminar la presión por parte de un grupo de usuarios. Los dispositivos son dos *Smartphones* de gama alta marca Sony y Samsung, donde previamente se les ha instalado la aplicación de nombre ***Prototype2.apk.***

Fase 2. Capacitación.

Antes de realizar la prueba, el usuario y/o usuarios se les pregunto acerca de su disposición para realizar la prueba, luego recibieron una pequeña introducción acerca del concepto de AR, junto con el contexto y las funcionalidades del escenario de interacción semántica, con el fin de que pudieran comprender cada escena de la aplicación y no generasen expectativas inesperadas ya que no era el fin de la aplicación a evaluar.

Fase 3. Prueba modo móvil.

El usuario sostuvo su dispositivo móvil con una mano y con la otra interactuo con el dispositivo, para realizar las funcionalidades básicas de cada escena a pedido del supervisor. El supervisor tomo atenta nota acerca de las preguntas y/o dificultades encontradas a lo largo del desarrollo de las tareas y brindo ayuda si alguna de estas no puede realizarse con éxito.

Posteriormente el usuario evaluó mediante el cuestionario algunos aspectos acerca de la facilidad de uso e intuición de la aplicación, y la asistencia prestada por el HUD.

El formulario utilizado consta de 10 afirmaciones tomadas y traducidas desde el test de usabilidad SUS [5], y 2 preguntas para evaluar la intuición de la interfaz, las afirmaciones son:

1) Creo que me gustaría utilizar esta aplicación frecuentemente.
2) Me pareció que la aplicación era innecesariamente compleja.
3) Me pareció que la aplicación era fácil de usar.
4) Creo que voy a necesitar la ayuda o la asistencia de un técnico o experto para ser capaz de utilizar esta aplicación.
5) Me pareció que las diversas funciones de la aplicación fueron bien integradas.

6) Pensé que habían demasiadas inconsistencias en la aplicación.
7) Me imagino que la mayoría de la gente aprendería rápidamente a utilizar esta aplicación.
8) Encontré que la aplicación era muy complicada de usar.
9) Me sentí con mucha confianza al utilizar la aplicación.
10) Tuve que aprender muchas cosas para comenzar a utilizar esta aplicación.
11) Creo que las representaciones virtuales de los objetos no tienen nada que ver con la realidad.
12) La interfaz parece saber lo que necesito para poder realizar las funciones de la aplicación.

El cuestionario se realizó a través de los Formularios de Google[9], ubicado en el siguiente enlace:

https://goo.gl/forms/p6DL835yuwTyuJA42

y se desplego a los sujetos de prueba como se observa en la figura 45.

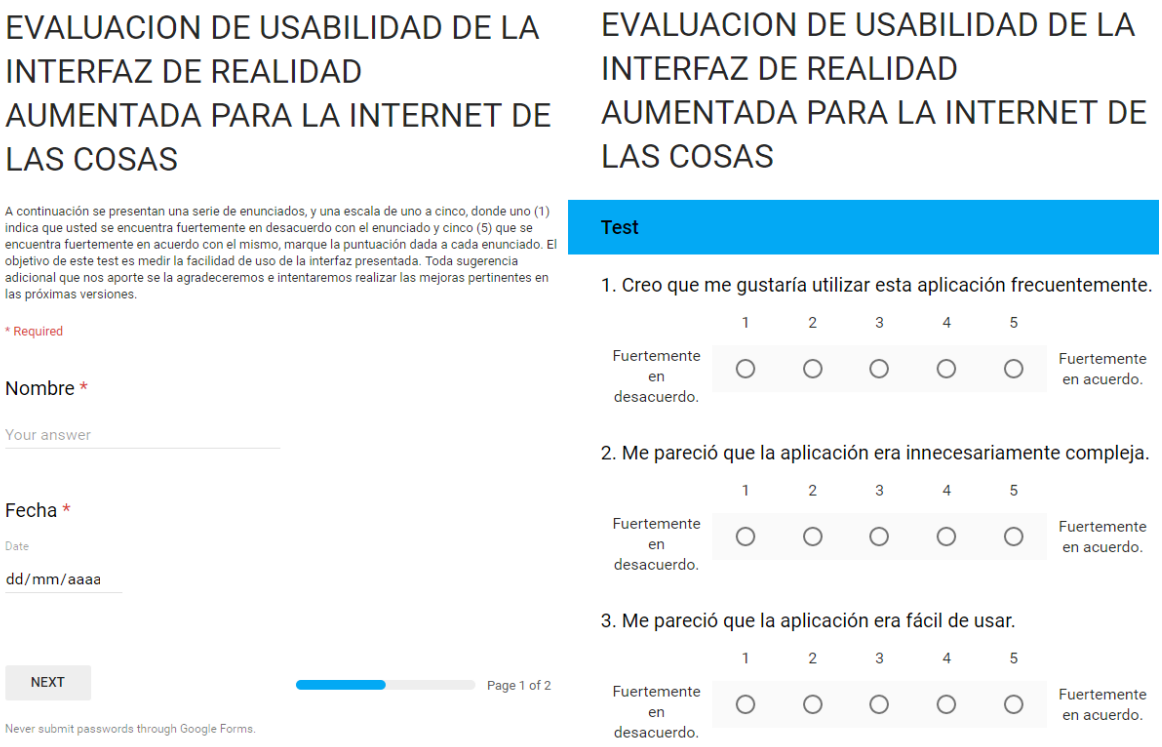

Figura 45. Evaluacion de usabilidad de la interfaz de AR para la IoT.

Los resultados por cada sujeto fueron:

[9] https://www.google.com/intl/es-419/forms/about/

Test SUS Sujeto 1		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	3	2
2	2	3
3	4	3
4	2	3
5	3	2
6	2	3
7	2	1
8	2	3
9	4	3
10	2	3
PUNTAJE SUS		65

Test SUS Sujeto 2		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	5	4
2	2	3
3	5	4
4	1	4
5	5	4
6	1	4
7	4	3
8	1	4
9	5	4
10	1	4
PUNTAJE SUS		95

Test SUS Sujeto 3		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	1	0
2	5	0
3	1	0
4	3	2
5	3	2
6	5	0
7	1	0
8	4	1
9	3	2
10	1	4
PUNTAJE SUS		27,5

Test SUS Sujeto 4		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	4	3
2	3	2
3	3	2
4	2	3
5	5	4
6	4	1
7	4	3
8	3	2
9	4	3
10	3	2
PUNTAJE SUS		62,5

Test SUS Sujeto 5		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	5	4
2	3	2
3	3	2
4	4	1
5	5	4
6	4	1
7	3	2
8	1	4
9	5	4
10	1	4
PUNTAJE SUS		70

Test SUS Sujeto 6		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	4	3
2	1	4
3	4	3
4	2	3
5	4	3
6	1	4
7	4	3
8	1	4
9	4	3
10	1	4
PUNTAJE SUS		85

Test SUS Sujeto 7		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	4	3
2	2	3
3	4	3
4	2	3
5	5	4
6	2	3
7	5	4
8	2	3
9	5	4
10	4	1
PUNTAJE SUS		77,5

Test SUS Sujeto 8		
PREGUNTAS	RESPUESTA	PUNTAJE SUS
1	4	3
2	4	1
3	3	2
4	3	2
5	5	4
6	3	2
7	4	3
8	3	2
9	4	3
10	3	2
PUNTAJE SUS		60

Figura 46. Resultados SUS por sujeto de prueba.