Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores
(Computer Engineering Academic Area)

Programa de Licenciatura en Ingeniería en Computadores
(Licentiate Degree Program in Computer Engineering)

Curso: CE5303 Introducción a Sistemas Embebidos
(Course: CE5303 - Embedded Systems Introduction)

Proyecto 1 - Sistema a la medida para el control y monitoreo de una casa inteligente por medio de servidor web

(Project 1 - Customized system for control and monitoring of a smart house through web server)

Realizado por:
(Made by:)
Arturo Salas Delgado – 201229804
Greivin Fallas Sánchez – 201036901
César Solís Valverde - 201217648

Profesor:
(Professor)
Jeferson González Gómez

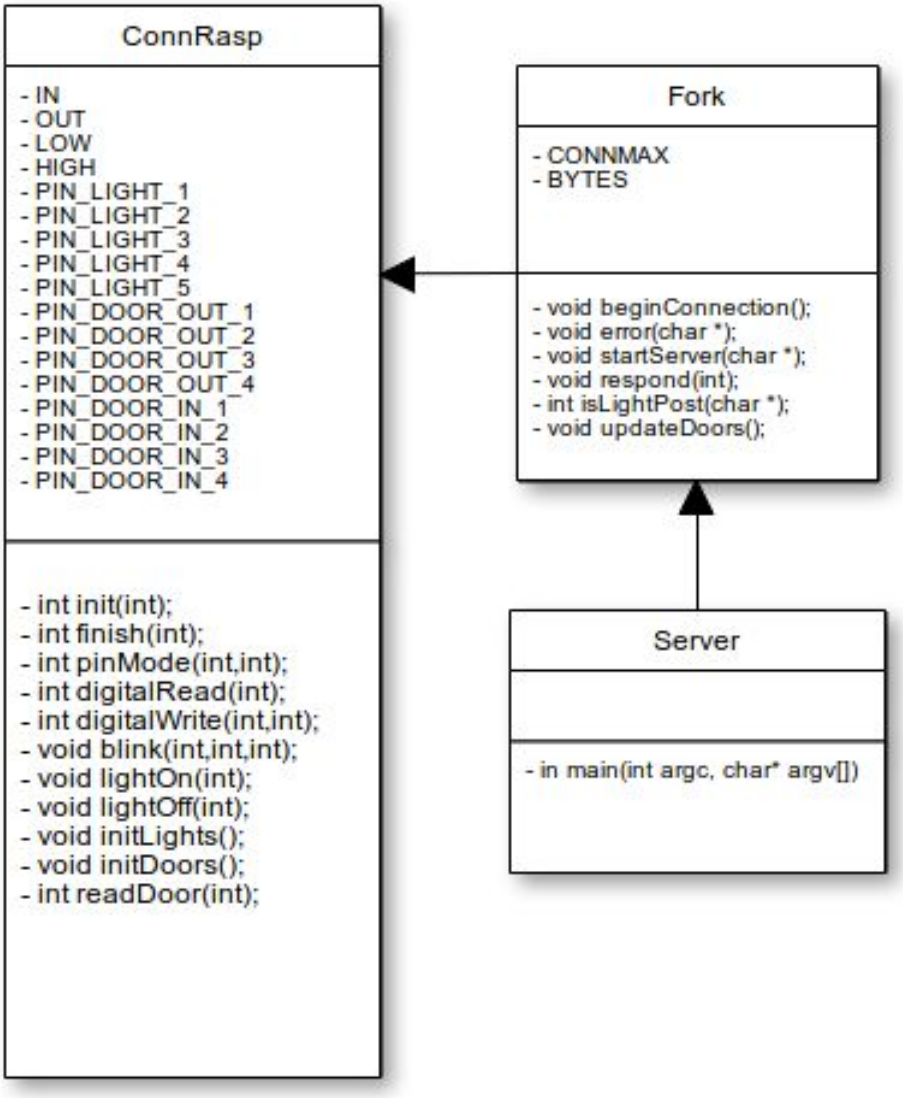
Fecha: Cartago, 15 de setiembre, 2017
(Date: Cartago, September 15, 2017)

Índice

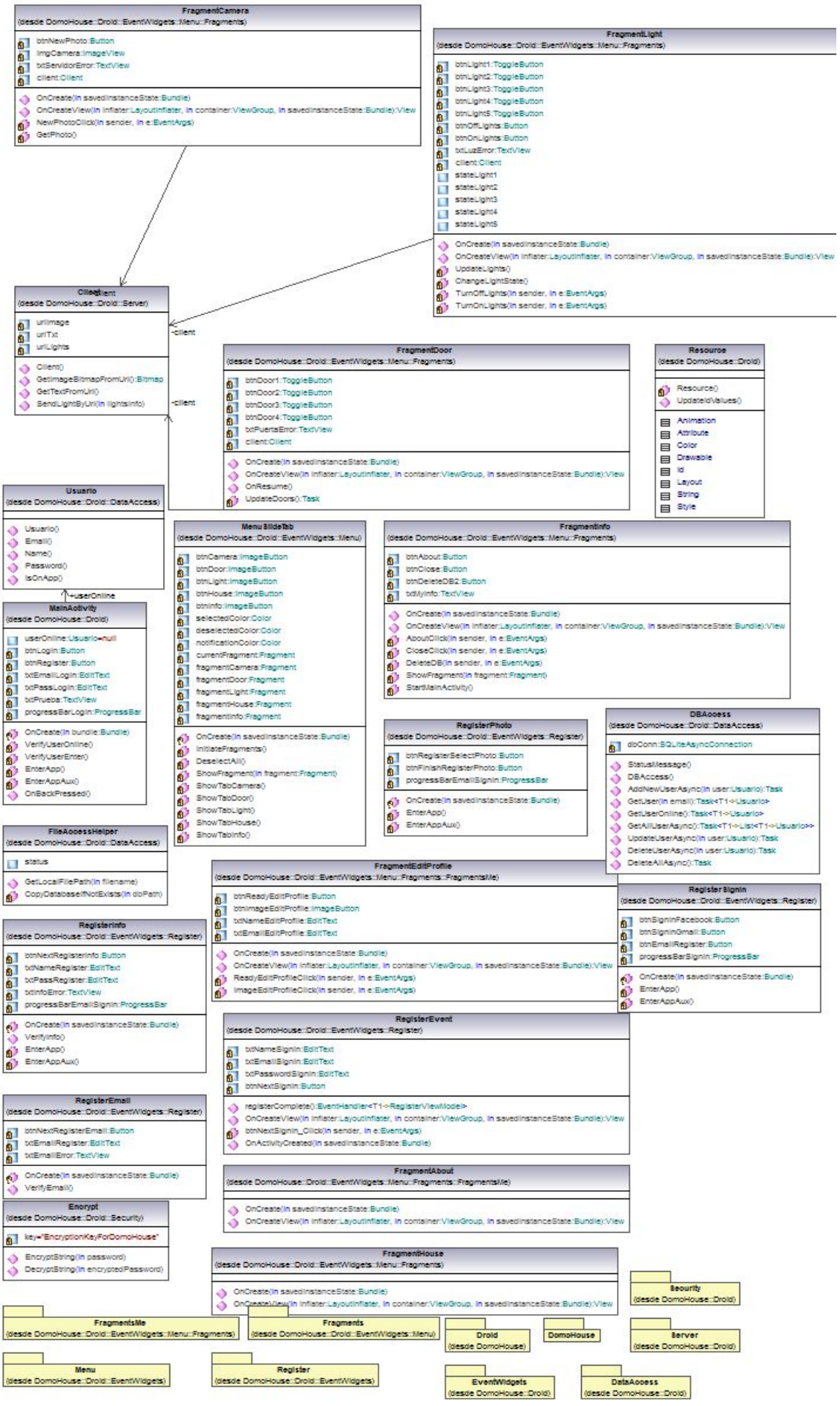
Diseño de Software

Diagramas de Clase

A continuación se muestra el diagramas de clases correspondiente al servidor y conexión mediante GPIO utilizada para la integración en la Raspberry Pi 2.

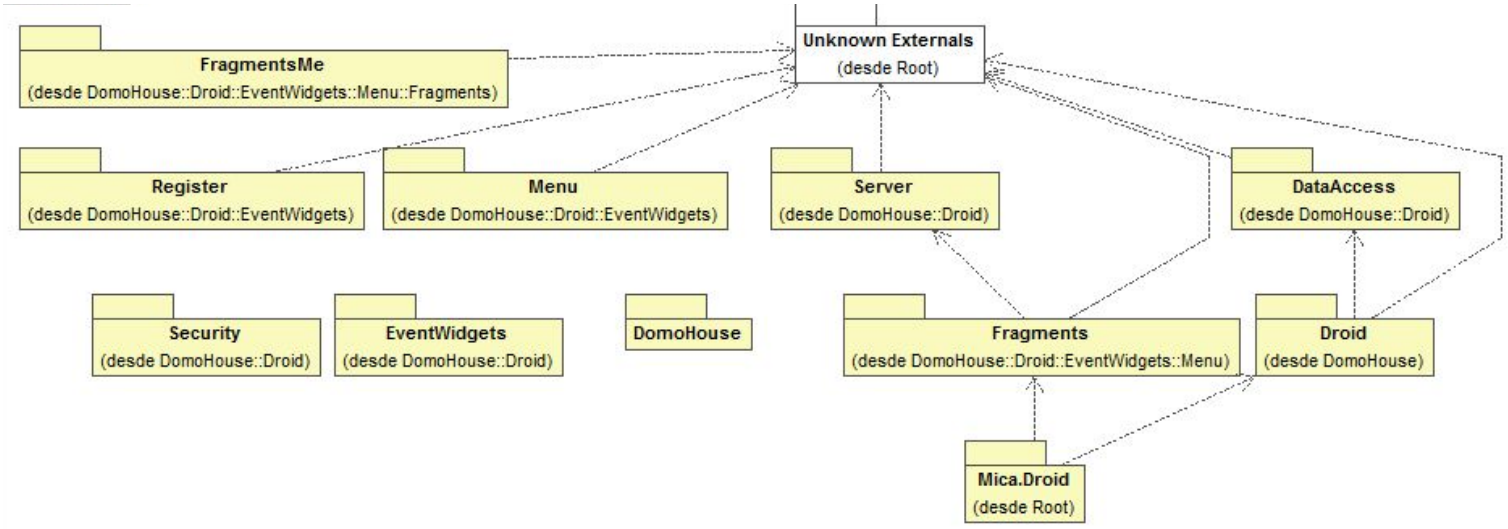


A continuación se muestra el diagrama de clases correspondiente a la aplicación DomoHouse. De no verse claramente, se puede descargar la imagen en el siguiente link: goo.gl/WBMkML


FragmentCamera
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments)
btnNewPhoto:Button
imgCamera:ImageView
txtServidorError:TextView
client:Client
OnCreate(In savedInstanceState:Bundle)
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
NewPhotoClick(In sender, In e:EventArgs)
GetPhoto()
FragmentLight
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments)
btnLight1:ToggleButton
btnLight2:ToggleButton
btnLight3:ToggleButton
btnLight4:ToggleButton
btnLight5:ToggleButton
btnOffLights:Button
btnOnLights:Button
txtLuzError:TextView
client:Client
stateLight1
stateLight2
stateLight3
stateLight4
stateLight5
OnCreate(In savedInstanceState:Bundle)
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
UpdateLights()
ChangeLightState()
TurnOffLights(In sender, In e:EventArgs)
TurnOnLights(In sender, In e:EventArgs)
Client
(desde DomoHouse::Droid::Server)
urlImage
urlTxt
urlLights
Client()
GetImageBitmapFromUrl():Bitmap
GetTextFromUrl()
SendLightByUrl(In lightsInfo)
-client
-client
FragmentDoor
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments)
btnDoor1:ToggleButton
btnDoor2:ToggleButton
btnDoor3:ToggleButton
btnDoor4:ToggleButton
txtPuertaError:TextView
client:Client
OnCreate(In savedInstanceState:Bundle)
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
OnResume()
UpdateDoors():Task
Resource
(desde DomoHouse::Droid)
Resource()
UpdateIdValues()
Animation
Attribute
Color
Drawable
Id
Layout
String
Style
Usuario
(desde DomoHouse::Droid::DataAccess)
Usuario()
Email()
Name()
Password()
IsOnApp()
+userOnline
MainActivity
(desde DomoHouse::Droid)
userOnline:Usuario=null
btnLogIn:Button
btnRegister:Button
txtEmailLogIn:EditText
txtPassLogIn:EditText
txtPrueba:TextView
progressBarLogIn:ProgressBar
OnCreate(In bundle:Bundle)
VerifyUserOnline()
VerifyUserEnter()
EnterApp()
EnterAppAux()
OnBackPressed()
MenuSlideTab
(desde DomoHouse::Droid::EventWidgets::Menu)
btnCamera:ImageButton
btnDoor:ImageButton
btnLight:ImageButton
btnHouse:ImageButton
btnInfo:ImageButton
selectedColor:Color
deselectedColor:Color
notificationColor:Color
currentFragment:Fragment
fragmentCamera:Fragment
fragmentDoor:Fragment
fragmentLight:Fragment
fragmentHouse:Fragment
fragmentInfo:Fragment
OnCreate(In savedInstanceState:Bundle)
InitiateFragments()
DeselectAll()
ShowFragment(In fragment:Fragment)
ShowTabCamera()
ShowTabDoor()
ShowTabLight()
ShowTabHouse()
ShowTabInfo()
FragmentInfo
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments)
btnAbout:Button
btnClose:Button
btnDeleteDB2:Button
txtMyInfo:TextView
OnCreate(In savedInstanceState:Bundle)
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
AboutClick(In sender, In e:EventArgs)
CloseClick(In sender, In e:EventArgs)
DeleteDB(In sender, In e:EventArgs)
ShowFragment(In fragment:Fragment)
StartMainActivity()
RegisterPhoto
(desde DomoHouse::Droid::EventWidgets::Register)
btnRegisterSelectPhoto:Button
btnFinishRegisterPhoto:Button
progressBarEmailSignIn:ProgressBar
OnCreate(In savedInstanceState:Bundle)
EnterApp()
EnterAppAux()
DBAccess
(desde DomoHouse::Droid::DataAccess)
dbConn:SQLiteAsyncConnection
StatusMessage()
DBAccess()
AddNewUserAsync(In user:Usuario):Task
GetUser(In email):Task<T1->Usuario>
GetUserOnline():Task<T1->Usuario>
GetAllUserAsync():Task<T1->List<T1->Usuario>>
UpdateUserAsync(In user:Usuario):Task
DeleteUserAsync(In user:Usuario):Task
DeleteAllAsync():Task
FileAccessHelper
(desde DomoHouse::Droid::DataAccess)
status
GetLocalFilePath(In filename)
CopyDatabaseIfNotExists(In dbPath)
FragmentEditProfile
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments::FragmentsMe)
btnReadyEditProfile:Button
btnImageEditProfile:ImageButton
txtNameEditProfile:EditText
txtEmailEditProfile:EditText
OnCreate(In savedInstanceState:Bundle)
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
ReadyEditProfileClick(In sender, In e:EventArgs)
ImageEditProfileClick(In sender, In e:EventArgs)
RegisterSignIn
(desde DomoHouse::Droid::EventWidgets::Register)
btnSignInFacebook:Button
btnSignInGmail:Button
btnEmailRegister:Button
progressBarSignIn:ProgressBar
OnCreate(In savedInstanceState:Bundle)
EnterApp()
EnterAppAux()
RegisterInfo
(desde DomoHouse::Droid::EventWidgets::Register)
btnNextRegisterInfo:Button
txtNameRegister:EditText
txtPassRegister:EditText
txtInfoError:TextView
progressBarEmailSignIn:ProgressBar
OnCreate(In savedInstanceState:Bundle)
VerifyInfo()
EnterApp()
EnterAppAux()
RegisterEvent
(desde DomoHouse::Droid::EventWidgets::Register)
txtNameSignIn:EditText
txtEmailSignIn:EditText
txtPasswordSignIn:EditText
btnNextSignIn:Button
registerComplete():EventHandler<T1->RegisterViewModel>
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
btnNextSignIn_Click(In sender, In e:EventArgs)
OnActivityCreated(In savedInstanceState:Bundle)
RegisterEmail
(desde DomoHouse::Droid::EventWidgets::Register)
btnNextRegisterEmail:Button
txtEmailRegister:EditText
txtEmailError:TextView
OnCreate(In savedInstanceState:Bundle)
VerifyEmail()
FragmentAbout
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments::FragmentsMe)
OnCreate(In savedInstanceState:Bundle)
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
Encrypt
(desde DomoHouse::Droid::Security)
key="EncryptionKeyForDomoHouse"
EncryptString(In password)
DecryptString(In encryptedPassword)
FragmentHouse
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments)
OnCreate(In savedInstanceState:Bundle)
OnCreateView(In inflater:LayoutInflater, In container:ViewGroup, In savedInstanceState:Bundle):View
Security
(desde DomoHouse::Droid)
FragmentsMe
(desde DomoHouse::Droid::EventWidgets::Menu::Fragments)
Fragments
(desde DomoHouse::Droid::EventWidgets::Menu)
Droid
(desde DomoHouse)
DomoHouse
Server
(desde DomoHouse::Droid)
Menu
(desde DomoHouse::Droid::EventWidgets)
Register
(desde DomoHouse::Droid::EventWidgets)
EventWidgets
(desde DomoHouse::Droid)
DataAccess
(desde DomoHouse::Droid)

Diagramas UML

A continuación se muestra el diagrama de dependencia entre paquetes de la aplicación DomoHouse implementada:



Descripción de Métodos

A continuación se describen los métodos de las clases Fork y ConnRasp utilizadas para la implementación del sistema embebido en las Rapberry Pi 2.

1. ConnRasp.h - ConnRasp.c
    a. int init(int pin): este método se encarga de preparar el pin para una conexión, o sea, para poder escribir o leer bytes en ese pin.
    b. int finish(int pin): se encarga de finalizar la conexión con un pin.
    c. int pinMode(int pin, int mode): este método se encarga de preparar al pin para lctura o para escritura.
    d. int digitalRead(int pin): este método se encarga de leer un 0 o un 1 desde un pin, una vez que se inicializó la conexión con el mismo.
    e. int digitalWrite(int pin, int value): este método se encarga de escribir un 0 o un 1 a cierto pin, una vez que se inicializó la conexión.
    f. void blink(int pin, int frec, int time): este método se encarga de prender y apagar un led en cierto pin a cierta frecuencia.
    g. void lightOn(int pin): este método se encarga de prender un led ubicada en cierto pin. En sí, utiliza al método digitalWrite.
    h. void lightOff(int pin): este método se encarga de apagar un led ubicada en cierto pin. En sí, utiliza al método digitalWrite.
    i. int readDoor(int pin): este metodo se encarga de leer el valor de un push button, que representa el estado de una puerta. En sí, utiliza el método digitalRead.
    j. void initLights(): se encarga de inicializar todos los pins correspondientes a leds para poder escribir sobre ellos.
    k. void initDoors(): se encarga de inicializar todos los pins correspondientes a push buttons para poder leer su estado el algún momento.

2. Fork.h - Fork.c
    a. void startDomoHouse(): este método se encarga de inicializar todos los pins correspondientes a leds y push button que representan a luces y puertas respectivamente. Además, prepara la conexión del servidor.
    b. void beginConnection(): establece el puerto en el que se inicia el servidor, además, crea un archivo llamada casa.txt en donde se muestra toda la información de luces y puerta de la casa.
    c. void startServer(char *port): este método se encarga de iniciar el servidor tipo fork en el puerto 8033, si hay error, bloquea el socket correspondiente.
    d. void respond(int n): este método se encarga de verificar si un cliente a establecido conexión con el servidor, si es así, verifica si es un GET o un POST para realizar el trabajo correspondiente y enviar bytes correspondientes a un archivo solicitado, en caso de ser necesario.
    e. int isLightPost(char *input): si el cliente necesita editar la información de las luces de la casa, este método se encarga de abrir el archivo casa.txt para

modificar el estado de las luces e interpretar los datos para prender y/o apagar luces.

f. void updateDoors(): si el cliente necesita saber el estado de las puerta, este se encarga de leer el archivo casa.txt para ver el estado y devolver la información al cliente.

3. Server.c
    a. int main(): este método se encarga de invocar una conexión con el servidor. Es el método para iniciar la conexión con la casa.

Y en la siguiente parte se muestra la descripción de los métodos de la aplicación implementada:

1. MainActivity.cs
    a. private async void VerifyUserOnline(): este método verifica si algún usuario está autenticado con la app, ya que en el momento que se abre la app, inicia sesión con el usuario activo, si no, envía a la pantalla principal.
    b. private async void VerifyUserEnter(): método auxiliar para VerifyUserOnline, donde se obtiene el usuario en sesión.
    c. private void EnterApp(): Una vez que se hizo login o sign in, con este método se inicia la app DomoHouse.
    d. private void EnterAppAux(): Método auxiliar de EnterApp en donde se inicia una animación para el loggeo de un usuario.
2. RegisterEmail.cs
    a. public void VerifyEmail(): Verifica si se accesó un email en la aplicación para crear un nuevo usuario.
3. RegisterInfo.cs
    a. public async void VerifyInfo(): verifica si se accesó un nombre y una contraseña para crear un nuevo usuario.
    b. private void EnterApp(): Una vez que se hizo login o sign in, con este método se inicia la app DomoHouse.
    c. private void EnterAppAux(): Método auxiliar de EnterApp en donde se inicia una animación para el loggeo de un usuario.
4. MenuSlideTab.cs
    a. private void InitiateFragments(): inicia los 5 fragmentos de la aplicación. El fragmento de la foto, de la casa, de las luces, de las puertas y el de información.
    b. private void DeselectAll(): método que selecciona el fragmento elegido por el usuario, y deselecciona los demás.
    c. private void ShowFragment(Fragment fragment): método que inicializa un fragmento para mostrar en la aplicación.

d. private void ShowTabCamera(): método que muestra el fragmento correspondiente a la cámara.
e. private void ShowTabDoor(): método que muestra el fragmento correspondiente a las puertas.
f. private void ShowTabLight(): método que muestra el fragmento correspondiente a las luces.
g. private void ShowTabHouse(): método que muestra el fragmento correspondiente al mapa de la casa.
h. private void ShowTabInfo(): método que muestra el fragmento correspondiente a la información de usuario.

5. FragmentCamera.cs
    a. private void GetPhoto(): método que obtiene una foto del servidor mediante el uso de un cliente web.
6. FragmentDoor.cs
    a. async Task UpdateDoors(): método que actualiza la información de las puertas (activa o inactiva) mediante el uso de un cliente web.
7. FragmentInfo.cs
    a. void AboutClick(object sender, EventArgs e): método que abre un nuevo fragmento realacionado a la información de la aplicación.
    b. async void CloseClick(object sender, EventArgs e): método que se encarga de cerrar sesión del usuario actual.
    c. private void StartMainActivity(): una vez que se cerró sesión, este método se encarga de mostrar la vista principal de la aplicación.
8. FragmentLight.cs
    a. private void UpdateLights(): método encargado de actualizar la información actual de las luces mediante el uso de un cliente web.
    b. private void ChangeLightState(): método encargado de cambiar el estado de una luz mediante un cliente web.
    c. private void TurnOffLights(object sender, EventArgs e): método encargado de apagar todas las luces de la casa mediante el uso de un cliente web.
    d. private void TurnOnLights(object sender, EventArgs e): método encargado de encender todas las luces de la casa mediante el uso de un cliente web.
9. DBAccess.cs
    a. public async Task AddNewUserAsync(User user): método encargado de ingresar un nuevo usuario en la base de datos local.
    b. public async Task<Usuario> GetUser(string email): método que se encarga de obtener la información correspondiente a un usuario en específico desde la base de datos local.
    c. public async Task<Usuario> GetUserOnline(): método que se encarga de obtener el usuario en sesión de la aplicación, si es que existe.

d. public async Task<List<Usuario>> GetAllUserAsync(): método encargado de obtener todos los usuario existentes en la aplicación.
e. public async Task UpdateUserAsync(User user): método encargadao de actualizar la información de un usuario en la base de datos local.
f. public async Task DeleteUserAsync(User user): método encargado de eliminar un usuario en específico en la base de datos local.
g. public async Task DeleteAllAsync(): método encargado de eliminar todo lo que existe en la base de datos local.

10. FileAccessHelper.cs
    a. public static string GetLocalFilePath(string filename): método que obtiene el directorio donde se encuentra el archivo .db3 de la base de datos local.
    b. private static void CopyDatabaseIfNotExists(string dbPath): método que crea una nueva base de datos local en caso de que no exista.
11. Encrypt.cs
    a. public static string EncryptString(string password): método que encripta un texto, en este caso una contraseña, para guardarlo en la base de datos local para un nuevo usuario.
    b. public static string DecryptString(string encryptedPassword): método que desencripta la contraseña correspondiente a un usuario en específico para compararla con la ingresada en el sistema para un sign in.
12. Client.cs
    a. public Bitmap GetImageBitmapFromUrl(): método que hace un request al servidor para obtener el arreglo de bytes correspondiente a la imagen actual de la casa.
    b. public string GetTextFromUrl(): método que obtiene en texto la información de la casa, o sea, estado de luces y puertas.
    c. public void SendLightByUrl(string lightsInfo): método que envía al servidor un cambio en las luces de la casa.

Descripción de Bibliotecas

A continuación se describen las bibliotecas utilizadas en las clases ConnRasp y Fork, que son las dos clases fundamentales para la conexión con el sistema embebido.

1. stdio: por sus siglas en inglés, Standard Input-Output Header es la biblioteca que contiene macros y constantes para la programación en C para realizar operaciones de entrada y salida.
2. string: es una biblioteca que contiene operaciones para la manipulación de memoria.
3. stdlib: Standard Library es una biblioteca de propósito general. Puede encargarse de la gestión de memoria, control de procesos y entre muchas otras cosas.
4. unistd: es la biblioteca que provee acceso a las operaciones POSIX.
5. sys/types: es la librería que provee macros y definiciones de sistema operativo.
6. sys/stat: es la biblioteca que define la estructura de un dato retornado.
7. sys/socket: esta biblioteca provee definiciones relacionadas a sockets: sus tipos, direcciones, opciones de conexión, etc.
8. arpa/inet: es la biblioteca que contiene definiciones para las operaciones de internet.
9. netdb: esta biblioteca define la estructura hostent.
10. signal: esta biblioteca específica como un programa va a manejar señales mientras se ejecuta, para describir el comportamiento, por ejemplo, una división entre cero.
11. fcntl.h: esta biblioteca provee constantes simbólicas y macros para llamadas al sistema (System Calls).

A continuación se detallan las bibliotecas usadas en la aplicación implementada DomoHouse:

1. Android.App: es una conglomeracion de clases de alto nivel que representan el modelo de aplicación de Android.
2. Android.Content: es un grupo de clases especializadas para el acceso y transferencia de datos en el dispositivo.
3. Android.OS: provee de servicios de sistema básicos, transferencia de mensajes y procesos de comunicación en el dispositivo.
4. Android.Views: provee clases que se encargan del manejo de la pantalla y las interacciones con el usuario.
5. Android.Widget: Es un paquete de elementos gráficos para usar en la pantalla.
6. Java.Lang: provee interfaces fundamentales para el desarrollo del lenguaje Java.
7. System.Collections.Generic: permite crear colecciones de datos.
8. System: permite a la aplicación tener acceso a recursos del sistema como servicios.
9. System.IO:provee a la aplicación el acceso a archivos y datos de entrada del teclado.
10. System.Net: permite a la aplicación usar recursos web.

11. System.Linq: es una interface que permite crear bases de datos de tamaño reducido
12. System.Text: permite la manipulación de texto dentro de la aplicación.
13. System.Threading.Tasks: permite clases para implementar hilos dentro de la aplicación.
14. System.Security.Cryptography:ofrece métodos de cifrado a la aplicación, se utiliza el algoritmo SHA.
15. Android.Content.PM: permite el manejo de la informacion de paquetes de una aplicación como actividades, permisos y servicios.
16. Android.Runtime: permite a las aplicaciones Java tener una instancia de la clase Runtime que deja a la aplicación interactuar con el ambiente en el que se ejecuta.
17. Android.Graphics: provee de herramientas de bajo nivel que permite generar gráficos directamente en la pantalla.
18. Android.Support.V7.App: provee la interacción de la barra de acciones.
19. Android.Support.V4.App.Fragment: permite que las aplicaciones se ejecuten sobre versiones superiores a Android 3.0.

Descripción de APIs

Android API 23: Las API son un grupo de funciones que puede brindar un determinado software de desarrollo de aplicaciones. Para Android las múltiples API que posee se asocian con su nivel. El mayor nivel corresponde a los dispositivos más recientes y provee mayores funciones mientras que el menor nivel es para los dispositivos más antiguos con menores funciones.

En el desarrollo de la aplicación se utilizó la API 23 de Android -conocida como API M- la cual provee una gran cantidad de funciones que permiten la correcta ejecución de la aplicación deseada. A nivel de mercado la API M es la que más dispositivos posee por lo tanto se determinó el uso de la API 23 para brindar a nuestra aplicación un mayor espectro de dispositivos compatibles.

Una API menor brinda menores funciones que pueden comprometer la correcta funcionalidad del software mientras que una API mayor a la 23 acota el espectro de dispositivos compatibles.

Para las API de Android se pueden utilizar niveles de API de un determinado valor y los niveles mayores permiten a la aplicación ejecutarse correctamente, es decir, la aplicación desarrollada de nivel 23 se puede ejecutar en dispositivos con niveles de API 23 o mayores, por ejemplo la API 26.

Requisitos de Software

Para el presente proyecto se requiere del desarrollo de un sistema embebido que debe de comunicarse con dispositivos móviles.

Para el proyecto se requiere que se cumplan las siguientes características: se debe desarrollar un servidor web mínimo embebido en la Raspberry que se encargue de controlar diferentes señales y sensores (switches o pulsadores).

Entre las diferentes señales se tiene que cumplir con al menos controlar 5 luces leds y 4 pulsadores. El control de las señales se debe realizar a través de una aplicación móvil que debe tener control de acceso por medio de usuario y contraseña con protocolo de seguridad, el usuario debe poder registrarse en la aplicación para acceder a manipular las señales.

El servidor web se debe mantener supervisando el estado de los pulsadores para detectar cambios, estos pulsadores representan puertas que se mostrarán en la aplicación como puertas abiertas o cerradas según el estado que detecte el servidor web.

El sistema embebido debe tener la capacidad de tomar fotografías y enviarlas a través del servidor web a la aplicación. Desde el dispositivo móvil se debe enviar comandos que el servidor debe responder, entre los comandos se encuentran: encender una o varias luces, encender o apagar todas las luces simultáneamente, supervisar el estado de las puertas y tomar una foto.

La biblioteca que se va a implementar para manipular las señales debe ser creada por el grupo de trabajo y debe ser dinámica. Para esta biblioteca se requiere del uso de toolchains que generen los archivos necesarios que va a requerir el ejecutable de la aplicación servidor web.

El sistema operativo de la Raspberry se debe crear con Yocto Project, de ser necesario se pueden agregar los paquetes de programas que se requieran mientras exista justificación. Para tomar las fotografías se requirió del programa fswebcam que se encarga de manipular cámaras web desde un terminal.

Desde el sistema operativo se requiere que al iniciar arranque el servidor sin la intervención de ningún usuario. El servidor debe mantenerse activo recibiendo peticiones de múltiples usuarios desde múltiples dispositivos móviles.

Todo el código debe ser empaquetado y los ejecutables deben de ser generados por medio de compilación cruzada utilizando Autotools y el toolchain de Yocto 2.3, arm-poky-linux.

Para efectos de presentación se requiere de un modelo de una casa que permita mostrar las funciones tanto del sistema embebido como de la aplicación.

Estado del software

Código	Requisito	Estado (0 - 5)
REQ01	Servidor web	5
REQ02	Aplicación móvil	5
REQ03	Biblioteca para pines propia	5
REQ04	Luces desde aplicación	5
REQ05	Supervisión de puertas desde aplicación	5
REQ06	Captura de imágenes desde aplicación	5
REQ07	Aplicación inicia con el booteo	5
REQ08	Sistema operativo creado con Yocto	5
REQ09	Compilación cruzada con Autotools	5

Metodología de Diseño de Sistema

A continuación se hará una explicación de la metodología utilizada en el proyecto, donde se involucra el análisis del problema, la investigación respectiva,algunas propuestas de diseño y la comparación y evaluación de tales propuestas .

Metodología Scrum

La metodología de diseño utilizada implica la división en módulos (compilación, ejecución e interfaz con el usuario), donde el comportamiento colectivo deseado de los sistemas emerge tras una serie de iteraciones de los componentes individuales. En esta, se pretende la reunión de diferentes sistemas (en este caso, módulos) para formar uno general (la aplicación deseada para el proyecto). Así, los módulos son especificados y separados según la conexión que conlleva cada una de las clases, donde cada paquete creado debe ser probado (es en cierta manera funcional) por separado para lograr un resultado deseado, el cual posteriormente será adjuntado al sistema final.

Por otro lado se establece no obtener todas las especificaciones desde un principio sino por medio de pequeños sprints (parecido a la metodología Scrum, véase la Figura 1, para lograr aportes pequeños y funcionales a ser transferidos a todo el sistema; es decir, cada vez se va trabajando de partes más pequeñas a partes más grandes. Así pues, se toma la decisión de usar la metodología mencionada aunque se sabe de sus desventajas como al trabajar en módulos más pequeños se pueden encontrar problemas al momento de unirlos al sistema general. Y también, buscar disminuir la documentación, obteniendo sólo lo necesario para desarrollar cada uno de los pequeños Sprints.

Algunas ventajas de las metodología Scrum son:

- Busca generar más y mejor software en menos tiempo.
- No genera tanta documentación.
- Usado no solo en software sino en proyectos que requieren flexibilidad y agilidad durante la construcción de los productos.
- Permite la construcción de proyectos de manera iterativa.
- Transparencia con respecto a los objetivos, avances y tiempos de entrega en el proyecto.

Análisis del Problema

El campo tecnológico ha mostrado un crecimiento en el desarrollo de aplicaciones que faciliten el diario vivir de la humanidad en general, los sistemas embebidos han contribuido en gran parte este campo dando solución a problemas específicos y de interés para situaciones de necesidad y/o comodidad de las personas.

El proyecto se centra en el control y monitoreo de una casa inteligente, cabe destacar que sistemas de domótica son conocidos e implementados de diversas formas en la actualidad, sin embargo, estos están asociados a un elevado costo monetario, parte del objetivo de este trabajo es dedicar al diseño una solución no solo óptima, sino que dentro de las

capacidades del bolsillo del promedio de las personas. El sistema ayuda a que el dueño del hogar pueda hacer uso razonable de sus recursos al poder manejar luces de las distintas habitaciones, además permite un buen control y monitoreo al poder detectar puertas abiertas y cerradas; claramente el punto a destacar es la captura de imágenes ya que con esto se tiene un control más delicado para el usuario ya que esta característica puede ser asociada al uso del timbre o a alguna otra solicitud expresa del usuario. Un protocolo de seguridad le adiciona privacidad y control a los dueños para protegerse de intercepciones de la comunicación como alteración de los accesos a la red del hogar.

Considerando la extensión del proyecto, tanto el servidor como el cliente móvil pueden ser extensibles para reconocer otros sensores en una casa tales como electrodomésticos u otra clase de sensores tales como sensores de gas, humo, agua, humedad con el fin de activar o desactivar dispositivos dentro de la casa. Para este desarrollo se cumplen con un panorama más básico con el fin de dar una validez de 'prueba de concepto' controlando los componentes descritos anteriormente. Cabe a destacar que el servidor web se usa como un servicio lo que quiere decir que arranca en el encendido, y manteniéndose la mayor parte del tiempo en funcionamiento y preparado para atender solicitudes desde su sistema operativo a la medida en el Raspberry Pi 2.

Por supuesto parte de la solución requiere de una cliente para el servidor el cual de forma muy intuitiva se implementa por medio de una aplicación móvil que es fácilmente adaptable para ser multiplataforma e incluso ser utilizada como un sitio web adaptativo; la simplicidad para agregar funcionalidades y hacer extensiones en el desarrollo se plantea desde el inicio ya que siempre se pueden agregar características al sistema y dar personalización según la casa inteligente que se adapte.

Para la metodología de desarrollo de forma general se trabaja por módulos como fue descrita en la descripción del sistema anteriormente [1] pero cabe destacar que como metodología de desarrollo para la parte de software también se aplica un proceso iterativo orientado en metodologías de desarrollo ágil y uso de un repositorio de control de código para el trabajo en conjunto del equipo.

Investigación Respectiva

La adquisición de los conocimientos requeridos para el desarrollo del proyecto ha sido mayormente a través de tutoriales vistos en clase centrados en la compilación cruzada y el manejo de un toolchain para dar soporte desde un computador personal al sistema embebido de la Raspberry pi 2.

Además, se obtuvo una introducción sobre el proyecto Yocto y la elaboración de imágenes a la medida [2] en Linux para sistemas empotrados. Entre los objetivos del proyecto se encuentra valorar la importancia de tener un sistema justo y reducido para únicamente las tareas que se desean realizar, también optimizar las gestiones del sistema para las tareas deseadas.

Para el control de señales en la Raspberry se requirió de elaborar una biblioteca personalizada, este es uno de los objetivos del proyecto por lo que requirió de investigación de las diferentes formas de manipular los pines, de ellas se eligió la interfaz sysfs que permite la manipulación directa de los recursos de la placa.[3]

Propuestas de Diseño

Para el caso del servidor, se tenían dos propuestas de implementación, una consistía en utilizar una arquitectura cliente - servidor, utilizando el lenguaje C para una integración más sencilla con la biblioteca GPIO; por otro lado, se tiene una propuesta de un servicio web RESTFUL en Javascript.

Otra propuesta de desarrollo fue el uso de módulos node para realizar la toma de fotografías, ante fswebcam, donde ambas toman una fotografía y la almacenan en memoria mediante un comando de sistema y una cámara web.

El proyecto solicita la realización de una aplicación para dispositivo móvil, para la cual el equipo decide usar la plataforma de Xamarin contra Ionic, que es una integración a Visual Studio para programación nativa de aplicaciones, utilizando el SDK de Android. Ionic es un framework basado en Angular, Javascript, HTML 5 y otras tecnologías web para generar aplicaciones en diferentes plataformas móviles.

Comparación y Evaluación de las Propuestas

Servidor: Se eligió la opción cliente-servidor en el lenguaje C con el fin de integrar fácilmente la biblioteca en c por desarrollar. La biblioteca que se debe desarrollar para la RaspberryPI es en lenguaje C y se basa en la interfaz Sysfs que provee la misma Raspberri, esta interfaz permite manipularlos pines directamente al manipular archivos en un directorio especial.

Con la biblioteca en C y el servidor en C limitado a transferir la mínima cantidad de datos se puede obtener un software a la medida de un tamaño muy reducido. Al utilizar C se pueden utilizar herramientas de desarrollo de sistemas embebidos conocidas como toolchains que genera ejecutables listos para transferir a la Raspberry, estos ejecutables con la ayuda de scripts conocidos en ambientes Linux como daemons permiten que el servidor se inicie desde el arranque, así se puede crear un sistema que es estable, mínimo y conforme a lo solicitado.

RESTFUL para ser ejecutado desde la Raspberry requiere de levantar un servicio web que se encargue de procesar las diferentes peticiones. Levantar el servicio web y el entorno requerido hacen que se requiera de más software del lado del sistema embebido, esto va en contra del objetivo de crear el sistema mínimo por lo tanto RESTFUL se descarta como opción.

Para cumplir con el cliente móvil se tiene la opción de desarrollo en Xamarin o Ionic. Entre estas opciones se eligió Xamarin debido a que esta herramienta ha sido utilizada en

proyectos previos, esto evita la curva de aprendizaje que se debe afrontar con Ionic debido a la escasa experiencia en su uso. Ambas propuestas pueden cumpli con el objetivo móvil y el desarrollo del sistema embebido.

Es importante mencionar que para crear el sistema más mínimo del sistema empotrado se va a requerir de transferir todas las funciones posibles al lado de la aplicación móvil. Xamarin e Ionic pueden cumplir con transferir lógica a la aplicación, la elección se determina por la curva de aprendizaje a evitar y por el tipo de tecnología que se utiliza en la elaboración de la aplicación.

En Xamarin a pesar de ser multiplataforma se puede generar aplicaciones con buen soporte nativo por parte del dispositivo, Ionic utiliza tecnologías web para generar la aplicación. Ionic puede generar aplicaciones visualmente cómodas y funcionales muy rápidamente, están hechas con el objetivo de consumir recursos web debido a que el soporte nativo no es tan bueno en un determinado dispositivo porque está orientado a generar aplicaciones multiplataforma.

Debido a los puntos anteriores y para aprovechar la funcionalidad nativa de los dispositivos se eligió Xamarin para Android, así se puede programar toda la lógica en la aplicación en lugar del sistema embebido dejando el software a la medida reducido al mínimo posible con todas las funcionalidades requeridas.

Herramientas de Ingeniería

Sublime Text 3.0

Editor de texto personalizado para múltiples lenguajes de programación. En este proyecto se utiliza para la modificación del código correspondiente al servidor y conexión GPIO con la Raspberry 2, para luego ser cross compilado.

Yocto 2.3 - Poky (Toolchain SDK)

Se utilizó la herramienta del proyecto de Yocto; además de Poky, la cual consiste en una colección de herramientas OpenEmbedded necesarias para construir una imagen mínima y crear una distribución de linux propia. [2]

Autotools

Es un conjunto de herramientas creadas para ayudar a crear paquetes de código fuente portable en múltiples sistemas Unix. Fue creado por el proyecto GNU. Es muy utilizado para el desarrollo de software libre.

EVM Raspberry Pi 2

Esta herramienta se encarga de controlar todas las señales de la casa, tanto luces, como puertas. Además, se encarga de tomar fotografías a petición del usuario de la aplicación. Sobre ella, se realizó la instalación de una imagen a la medida de Yocto, junto con las capas de personalización de node y fswebcam.

Visual Studio 2015: Xamarin

Xamarin es una plataforma integrada en Visual Studio especialmente para desarrollar aplicaciones nativas para Android, IOS y Windows Phone. Utiliza C# y XAML para desarrollar aplicaciones.

Adobe Illustrator 2017

Es un editor de gráficos vectoriales creado por la empresa Adobe Systems. Es una herramienta utilizada para la creación de la casa mediante corte láser. Contiene opciones creativas, acceso fácil a las herramientas y una gran versatilidad para producir gráficos.

Bibliografía

[1] H. Miguel. (2007). Diseño Modular. Laboratorio de Diseño Lógico. Tecnológico de Costa Rica.

[2] S. Rifenbark. (2017). Yocto Project Mega Manual. [En linea] Disponible en: http://www.yoctoproject.org/docs/2.3.1/mega-manual/mega-manual.html

[3] elinux.org. (2017). RPi GPIO Code Samples. [En línea] Disponible en: http://elinux.org/RPi_GPIO_Code_Samples