Smart Parking

Manejo de recursos escasos

Marcelo Bustos Ramírez.

Sebastián Robles González.

Cristian Vidal Sepúlveda.

Temuco, mayo 2018

Contenido

Introducción

En el mundo existen más de 1,1 billones de vehículos, en Chile más de 5 millones, en la Araucanía más de 220 mil (datos entregados por el instituto nacional de estadística en el año 2016), con el rápido avance de las ciudades y del crecimiento demográfico los espacios para construir estacionamientos se han reducido radicalmente y la demanda de estos crece exponencialmente cada año. Por esta razón se vuelve cada vez más necesario un sistema que sea capaz de optimizar y facilitar su manejo.

Con esto en mente nuestro equipo de trabajo busca desarrollar un sistema que permita detectar y manejar el estado de los espacios de un estacionamiento y con ello facilitar su gestión, navegabilidad además de reducir el tiempo invertido en estacionar. Todo esto con el fin de optimizar el uso de este recurso cada vez más escaso en la sociedad actual.

Para este prototipo nos basamos principalmente en tecnologías de Java y Arduino, además de herramientas de diseño, para generar las primeras propuestas de interfaces gráficas, y técnicas que permitan generar u organizar calendarios de trabajo y asignar responsabilidades. Adjunto a este informe se entrega un prototipo básico orientado a objeto del proyecto y nuestras impresiones de éste.

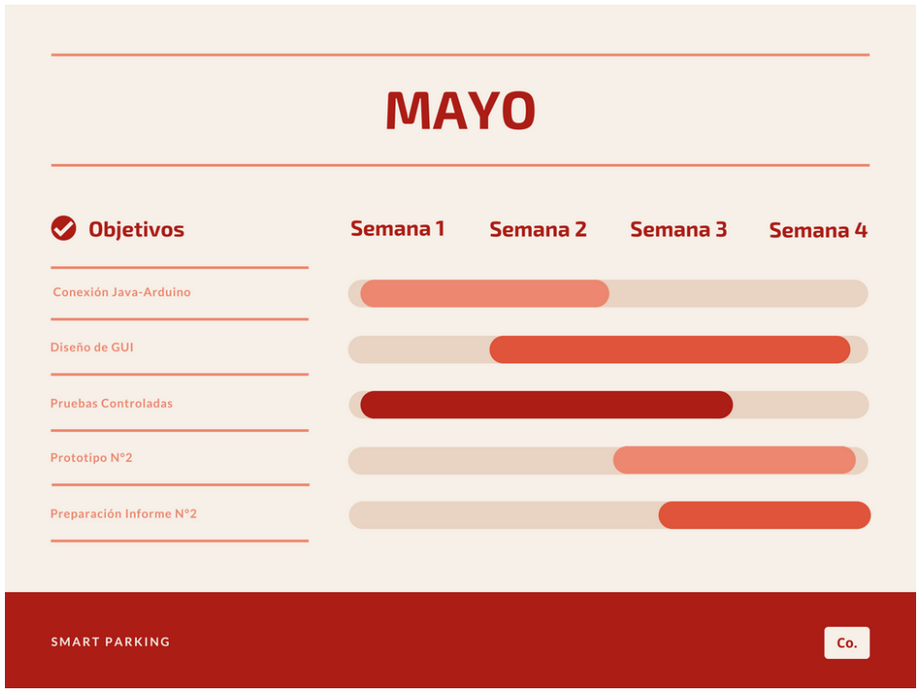
Debido a las correcciones del informe 1 la sección “Módulos y métodos” fue reemplazada por la sección documentación. Además, se agregó un paquete encargado de la recolección de datos en la sección “Gestión de Datos”. También se incorporaron esquemas para representar el flujo de información y funcionamiento del hardware.

Cambios en la planificación

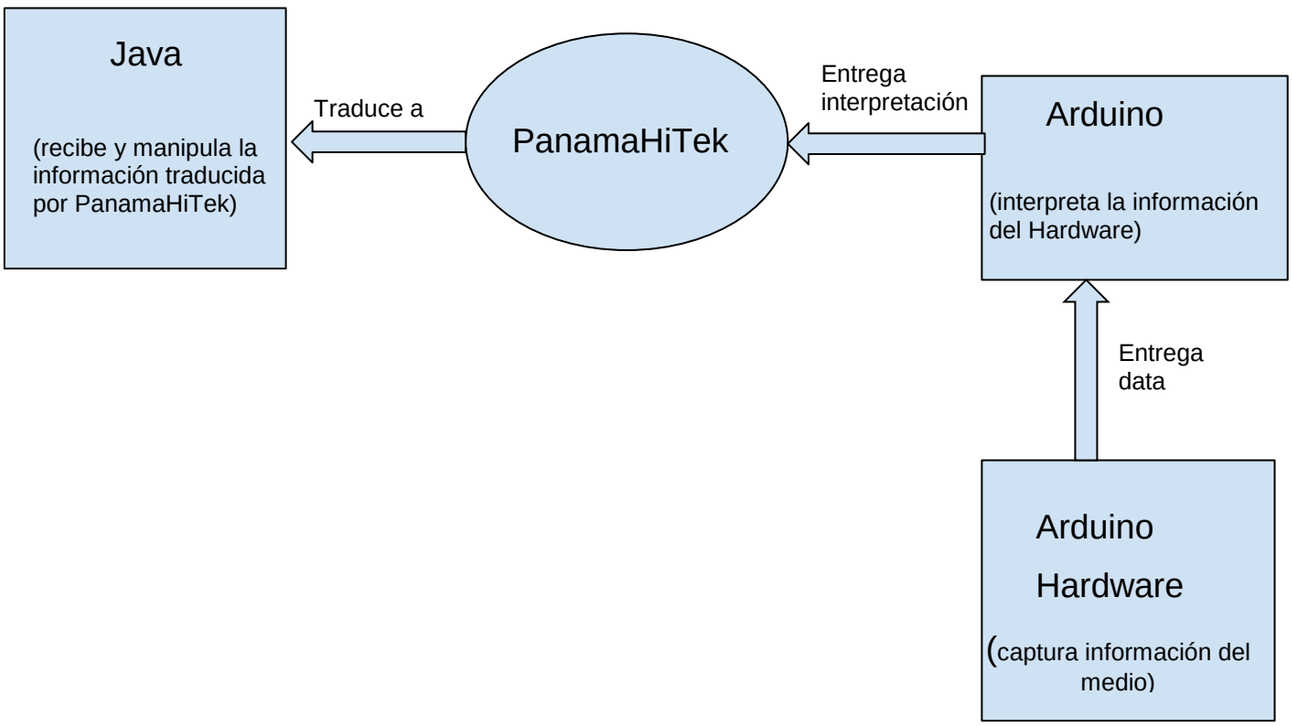
La planificación sufrió cambios debido a problemas estableciendo la conexión entre Arduino y Java, retrasando el proyecto en dos semanas ya que la conexión de estas tecnologías es parte principal del proyecto, por lo que la nueva planificación tiene menos tiempo de trabajo en ciertos aspectos como diseño para cumplir con los plazos requeridos.

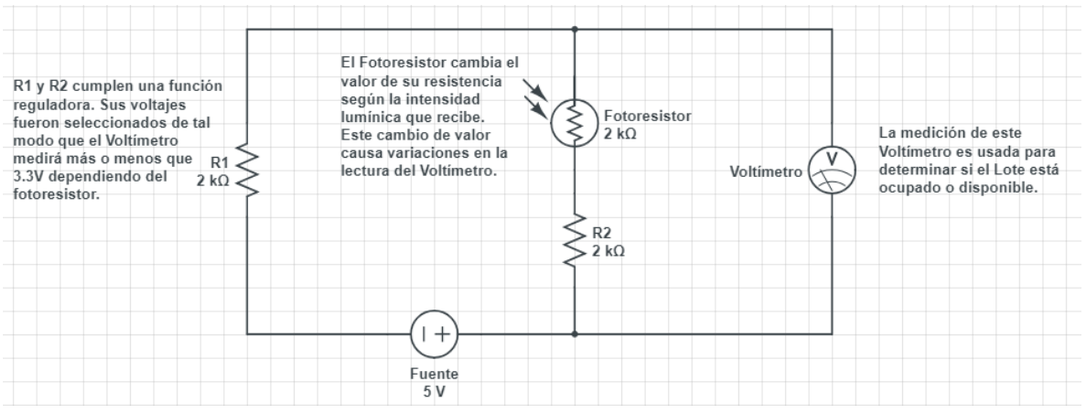
Cabe mencionar que actualmente nos encontramos terminando nuestro primer diseño de GUI y prontos a iniciar la incorporación del manejo de datos.

Nuevo Plan de Trabajo



Flujo de Información y Funcionamiento del Hardware





Documentación

Clase	Atributos	Metodos
SmartParking(Main)	ninguno	main():void
Lote implements cambioEstado	-estado:boolean -uso:String -listeners:ArrayList<>	+getEstado():boolean +setEstado(entrada:int):void +addEventListener(ChangeEvent listener):void +enEstado(ChangeEvent cv):void +enOcupado(ChangeEvent cv):void -triggerEnEstadoEvent():void -triggerOnEstadoEvent():void
Estacionamiento	ninguno	+getColor(estado:boolean):int -actualizarEstado(int pos, int valorConexion):void
ChangeEvent extends EventObject	lote:Lote	ChangeEvent(source:Object, _lote :Lote)
Interface cambioEstado extends EventListener	ninguno	enEstado(cv:ChangeEvent) enOcupado(cv:hangeEvent)
conexionArduino	-puerto:String +listener:SerialPortEventListener	+serialEvent:void() importa PanamaHitek_Arduino arduinoRx():void isMesaggeAvailable():boolean receiveData()
Data	ninguno	manejoData():void

getEstado en clase Lote entrega un booleano, verdadero si está disponible, falso si está ocupado.

setEstado en clase Lote permite a la clase estacionamiento cambiar el atributo estado de Lote.

El resto de los métodos en Lote permiten detectar cambios en el estado de este para enviarlos a la clase manejo de datos para su posterior almacenamiento.

La clase ChangeEvent envía un event cuando hay cambios en Lote.

Estacionamiento getColor procesa el estado de un lote y lo en base a este genera un int que indica el color que tendrá el espacio correspondiente al lugar del Lote en ventana.

Estacionamiento actualizarEstado se encarga de manejar el estado de los Lotes según los datos que reciba desde valorConexion.

conexionArduino SerialEvent recibe los datos captados por Arduino y los envía a Estacionamiento para su procesamiento.

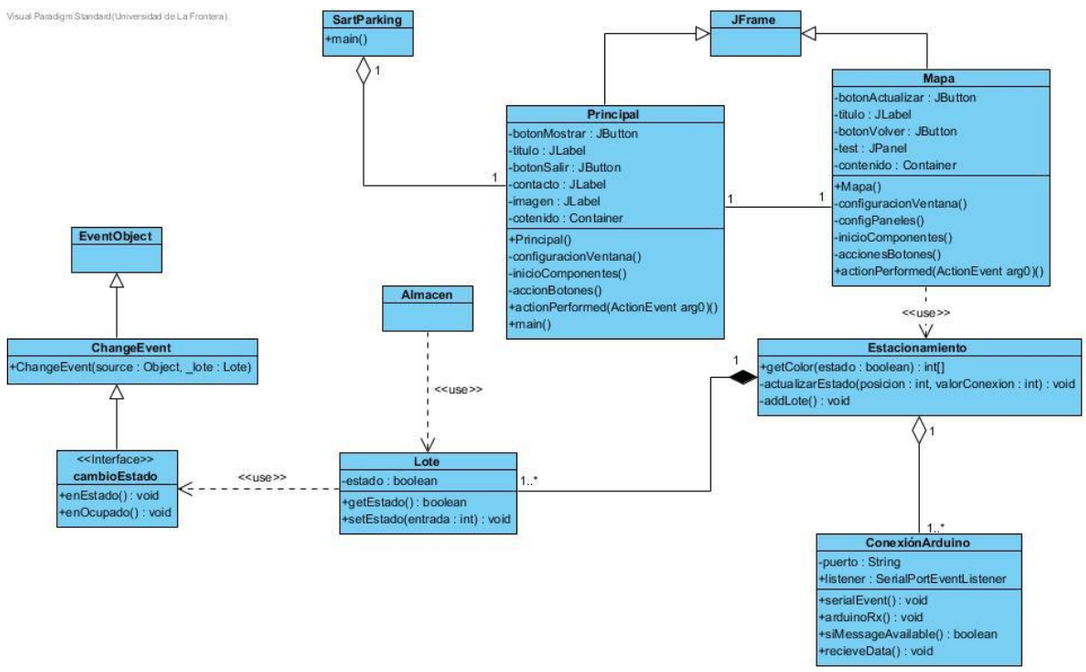
conexionArduino arduinoRx inicia en enlace entre java y Arduino, permite recibir los datos.

conexionArduino isMessageAvailabe retorna un booleano, verdadero si se recibe información desde Arduino y falso en caso contrario.

conexionArduino receiveData muestra o almacena los datos recibidos.

Data ManejoData guarda los datos recibidos y la fecha en un archivo txt (aún en fase de implementación).

Modelo UML


Visual Paradigm Standard (Universidad de La Frontera)
SartParking
+main()
JFrame
Mapa
-botonActualizar : JButton
-titulo : JLabel
-botonVolver : JButton
-test : JPanel
-contenido : Container
+Mapa()
-configuracionVentana()
-configPaneles()
-inicioComponentes()
-accionesBotones()
+actionPerformed(ActionEvent arg0)()
Principal
-botonMostrar : JButton
-titulo : JLabel
-botonSalir : JButton
-contacto : JLabel
-imagen : JLabel
-cotenido : Container
+Principal()
-configuracionVentana()
-inicioComponentes()
-accionBotones()
+actionPerformed(ActionEvent arg0)()
+main()
EventObject
Almacen
ChangeEvent
+ChangeEvent(source : Object, _lote : Lote)
Estacionamiento
+getColor(estado : boolean) : int[]
-actualizarEstado(posicion : int, valorConexion : int) : void
-addLote() : void
<<Interface>>
cambioEstado
+enEstado() : void
+enOcupado() : void
Lote
-estado : boolean
+getEstado() : boolean
+setEstado(entrada : int) : void
ConexiónArduino
-puerto : String
+listener : SerialPortEventListener
+serialEvent() : void
+arduinoRx() : void
+siMessageAvailable() : boolean
+recieveData() : void
<<use>>
<<use>>
<<use>>
1
1
1
1
1
1
1..*
1..*

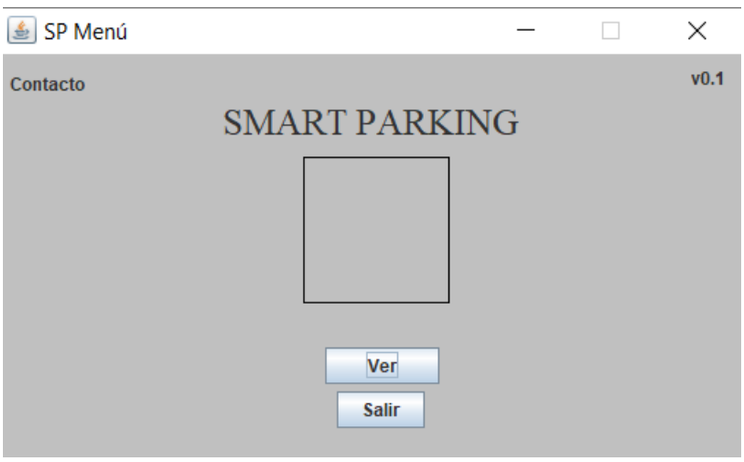
Diseño de GUI

Menú Principal o Primera Ventana

Esta va a ser la ventana que el usuario verá al ejecutar el programa, cumple la función de “portada” y de puente para la siguiente ventana, además incorporará un botón para ver los correos electrónicos de los integrantes del grupo.

Diseño Inicial

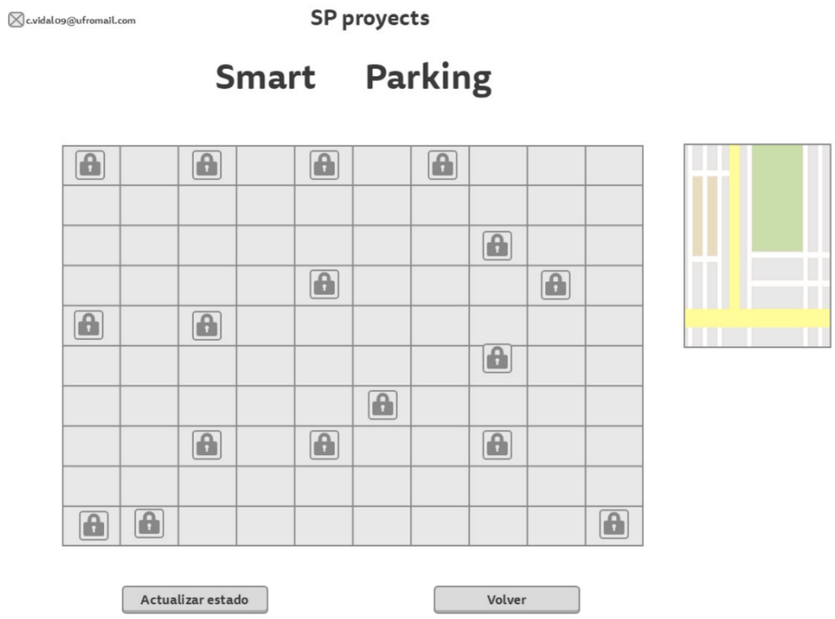
Primer prototipo hecho en java



Estacionamiento o Segunda Ventana

En esta ventana se muestra un estacionamiento y el estado de todos sus espacios ó lotes y la **disponibilidad** de estos**,** la cual es representada con un color, verde para “Disponible” y rojo para “Ocupado”.

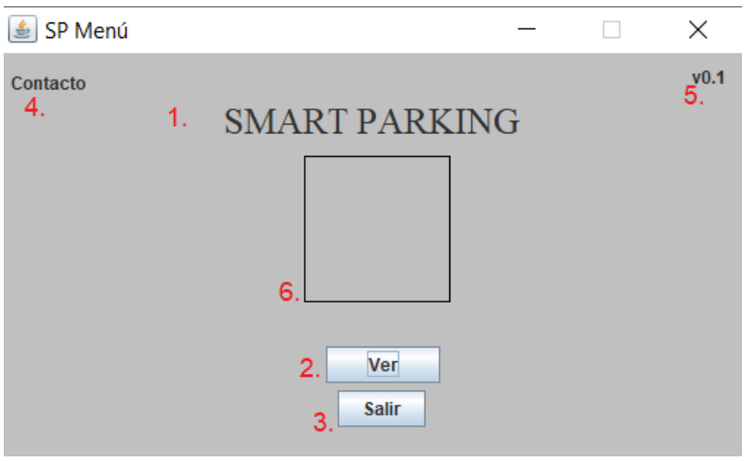
Diseño Inicial



Primer prototipo hecho en java

Mapa de Navegación

Ventana Principal



Componentes y sus funciones

1. JLabel para mostrar el nombre del producto.
2. **JButton para ver el estacionamiento**, al dar clic a este botón esta ventana se cierra y se abre la segunda ventana.
3. JButton para cerrar la aplicación, al dar clic a este botón se termina todo proceso (puede que este botón se quite en futuras actualizaciones).
4. JLabel para un espacio reservado para colocar un botón para ver los correos electrónicos.
5. JLabel para mostrar la versión del programa.
6. JLabel para dejar un espacio para poner una imagen.

Segunda Ventana

Componentes y sus funciones

1. **JButton para volver a la primera ventana**, al presionar este botón esta ventana se cierra y se vuelve a la “portada” de la aplicación.
2. JPanel para representar el estado de un espacio del estacionamiento, el color del JPanel cambiará según su estado.
3. JLabel para mostrar el nombre del producto.
4. **JButton para actualizar el estado del espacio.**

Datos a Gestionar

Como se mencionó en los cambios, se agregará una funcionalidad que guarde algunos datos relevantes.

El archivo que manejará será del tipo txt, con 3 columnas la primera con el sensor desde el cual llegan los datos, la segunda con la fecha y hora en la que el espacio se encuentra ocupado y la tercera columna la fecha en la que vuelve a estar disponible.

Habrá una sola clase que se encargue de gestionar los datos, ésta será la clase “ManejoDatos” perteneciente al paquete manejodedatos. Para ello se generarán cambios en la clase lote, además de incorporar una Interface y un EventListener con el fin de detectar los cambios en la variable estado. Cada vez que el estado de un Lote pasa a “Ocupado” uno de los eventListener agrega el valor de la fecha y del objeto que lo envía a un String y cuando ese mismo objeto regresa a “Disponible”, el segundo EventListener agrega la fecha de salida y envía los datos a la clase ManejoDatos para ser guardados en el txt y luego limpia la variable para una nueva ocurrencia del evento.

Conclusión

En conclusión, se han hecho correcciones basadas en nuestros resultados del informe N°1, además de resolver nuestro inconveniente más grande, establecer la conexión entre Arduino y Java. Sin embargo, dado nuestro retraso de dos semanas, una replanificación fue necesaria para cumplir con los plazos estipulados al comienzo del proyecto.