PROYECTO:

Replicant: Interfaz para la interoperabilidad de dispositivos embebidos con capacidad de conexión a redes de datos.

Autor: Juan Gregorio Regalado Pacheco
Consultor: Víctor Carceler Hontoria
Fecha: 10 de Junio de 2012

Índice de contenido

Plan de proyecto.

Objetivos generales del proyecto

Durante los últimos tiempos hemos asistido a la aparición de un nuevo concepto relacionado con los sistemas embebidos con capacidades de comunicación a través de redes de datos. Este concepto se ha denominado, “The Internet of Things”.

La idea que subyace bajo este nuevo término es la de permitir que objetos indentificados de manera unívoca, en la mayoría de los casos utilizando etiquetas RFID, puedan ser detectados e interactuar entre sí, generando una capa de información pública y disponible para cualquiera con un sistema lector adecuado.

A la luz de estos planteamientos parece lógico estudiar un escenario en el que objetos capaces de interactuar con el mundo físico a través de actuadores y sensores, puedan poner a disposición del público sus capacidades de una manera transparente al usuario final.

Para alcanzar esa meta, habría que diseñar un marco de trabajo y unas especificaciones mínimas que permitieran que los objetos presentes en el entorno próximo pudieran describir sus habilidades de interacción, poniéndose a disposición de cualquiera que solicitase sus servicios.

El objeto último de este proyecto consiste en establecer una interfaz que permita que un dispositivo embebido cualquiera, con capacidad para conectarse a una red de datos, pueda describir los actuadores y sensores de los que dispone y ponerlos al servicio de un usuario u otro sistema.

Como prueba de concepto se diseñará e implementará una aplicación para teléfonos móviles capaz de hacer uso de esta interfaz para controlar cualquier dispositivo que implemente dicha especificación al que pueda conectarse. También se diseñará y fabricará un dispositivo que pueda controlarse haciendo uso del marco de trabajo propuesto.

La denominación del proyecto proviene de los personajes de la película Blade Runner, en la que los “Replicantes” son máquinas que han tomado consciencia de sí mismas. Dada la naturaleza del proyecto parece una denominación adecuada, aunque quizá un tanto pretenciosa.

Resultados esperados

Al final del proyecto se espera contar con tres productos:

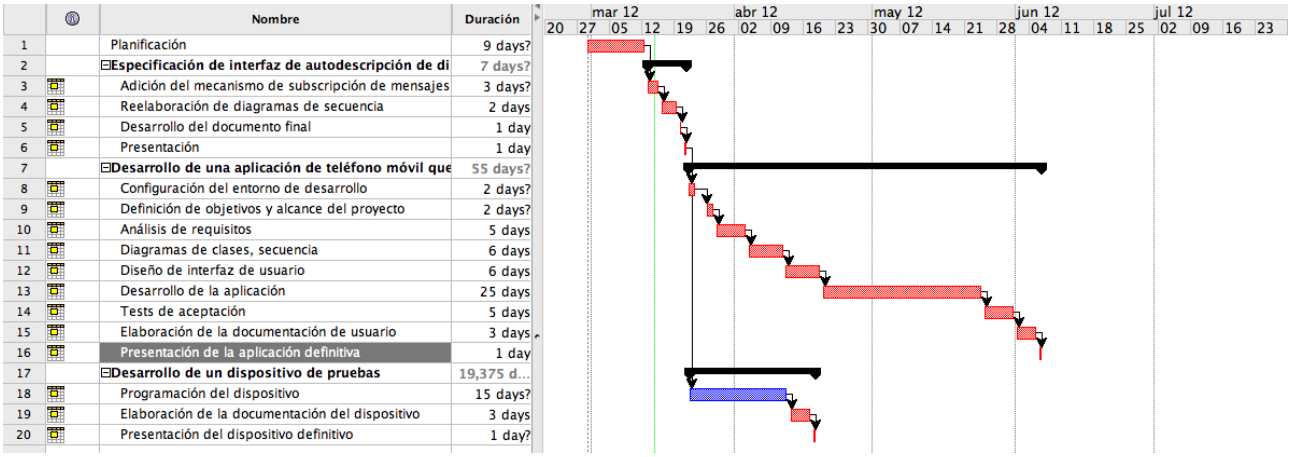
- Una especificación completa que permita a dispositivos conectados a una red describir sus capacidades y ponerse a disposición de usuarios u otros dispositivos que así lo soliciten.
- Una aplicación para teléfonos móviles con capacidad de conexión a redes de datos que pueda controlar dispositivos que implementen la interfaz anterior.
- Un dispositivo que implemente la interfaz anterior y que sirva como prueba de concepto.

Hitos

1. Presentación del plan de trabajo. Día 12 de marzo de 2012
2. Presentación de la especificación completa de la interfaz Replicant para la auto-descripción de dispositivos. Día 21 de marzo de 2012.
3. Presentación del dispositivo de pruebas funcional 18 de abril de 2012.
4. Presentación de la aplicación móvil para el control de dispositivos que implementan la

interfaz Replicant. Día 6 de junio de 2012.

5. Entrega final del proyecto. Día 12 de junio de 2012.



	Nombre	Duración
1	Planificación	9 days?
2	**Especificación de interfaz de autodescripción de di**	7 days?
3	Adición del mecanismo de subscripción de mensajes	3 days?
4	Reelaboración de diagramas de secuencia	2 days
5	Desarrollo del documento final	1 day
6	Presentación	1 day
7	**Desarrollo de una aplicación de teléfono móvil que**	55 days?
8	Configuración del entorno de desarrollo	2 days?
9	Definición de objetivos y alcance del proyecto	2 days?
10	Análisis de requisitos	5 days
11	Diagramas de clases, secuencia	6 days
12	Diseño de interfaz de usuario	6 days
13	Desarrollo de la aplicación	25 days
14	Tests de aceptación	5 days
15	Elaboración de la documentación de usuario	3 days
16	Presentación de la aplicación definitiva	1 day
17	**Desarrollo de un dispositivo de pruebas**	19,375 d...
18	Programación del dispositivo	15 days?
19	Elaboración de la documentación del dispositivo	3 days
20	Presentación del dispositivo definitivo	1 day?

Alcance y entregables

El proyecto se divide en tres grandes bloques.

1. **Diseño de la especificación completa de la interfaz Replicant:**

   La interfaz Replicant será la base sobre la que se construya el resto del proyecto, debe permitir que sistemas embebidos conectados a una red puedan autodescribirse y ponerse a disposición de usuarios u otros dispositivos.

   En el semestre anterior se acometió un análisis de proyectos afines al objetivo de éste, interfaces propuestas, así como iniciativas cuyo fin era interactuar mediante teléfonos móviles con dispositivos embebidos.

   También se estudiaron los conjuntos de propiedades que permiten describir un manipulador o robot móvil y sus conjuntos de sensores y actuadores.

   Durante este semestre se explorarán mecanismos de publicación/suscripción que permitan que sea el dispositivo el que inicie el envío de información a todos aquellos interlocutores que puedan estar interesados en la misma.

   Como producto de esta fase dispondremos de una propuesta de interfaz razonada así como un informe sobre la selección de la plataforma de desarrollo y la viabilidad de la implementación de la interfaz propuesta en dispositivos embebidos.

2. **Desarrollo de una aplicación para teléfonos móviles con capacidad de conexión a redes de datos que pueda controlar dispositivos que implementen la interfaz anterior.**

   En esta fase dedicaremos una parte de los recursos a la profundización en la plataforma seleccionada en la etapa anterior, de modo que a la salida de la misma tengamos una imagen clara de qué alternativas disponemos para acometer la implementación de la interfaz.

   Parte de los esfuerzos de esta fase se dedicarán al desarrollo de una interfaz de usuario que permita, de una manera ágil, gestionar los dispositivos detectados haciendo hincapié en la usabilidad de la misma.

Como producto final de esta etapa dispondremos de una aplicación para teléfonos móviles que permita detectar los dispositivos accesibles en la red, interactuar con sus actuadores y recibir datos de sus sensores para representarlos de manera adecuada.

3. **Desarrollo de un dispositivo que implemente la interfaz anterior y que sirva como prueba de concepto.**

   Durante el semestre anterior se llevó a cabo la construcción de la estructura física del dispositivo de pruebas. Sin embargo, no llegó a adaptarse a la primera propuesta de interfaz.

   El trabajo durante este semestre consistirá en implementar la interfaz sobre el microcontrolador del sistema de pruebas.

Roles y asignación de actividades

Siglas	Rol	Persona asignada
TS	Técnico de sistemas	Juan Gregorio Regalado Pacheco
IS	Ingeniería de sistemas	Juan Gregorio Regalado Pacheco
AF	Analista funcional	Juan Gregorio Regalado Pacheco
DE	Desarrollador	Juan Gregorio Regalado Pacheco
DEMB	Desarrollador dispositivos embebidos	Juan Gregorio Regalado Pacheco

Asignación de actividades y estimación del esfuerzo.

El personal asignado a este proyecto consta de una única persona (Juan Gregorio Regalado Pacheco). Los supuestos para la estimación de esfuerzos son los siguientes:

1. El personal trabajará, como máximo, 4 horas diarias durante 5 días a la semana, se considerarán festivos los sábados y domingos.
2. Durante la etapa de planificación, se dedicará la totalidad de la jornada laboral a la estimación de la temporalización, secuenciación de actividades y a la elaboración del plan de proyecto.
3. El siguiente paso en el desarrollo del proyecto consistirá en analizar las vías posibles para la adición de un mecanismo de publicación/suscripción en los dispositivos Replicant. Estas actividades se desarrollaran entre los días 13 y 21 de marzo.
4. Durante el periodo comprendido entre el 22 de marzo y el 18 de abril, el personal trabajará sobre dos actividades en paralelo.
   1. Por una parte se desarrollará toda la etapa de análisis y diseño de la aplicación móvil así como de su interfaz de usuario final.
   2. Por la otra se trabajará sobre la programación del dispositivo diseñado durante el semestre anterior.
5. Finalmente, durante el periodo del 19 de abril al 6 de junio, el personal se dedicará

plenamente al desarrollo y comprobación de la aplicación móvil.

Actividades	TS	IS	AF	DE	DEMB
Actividad 1: Especificación de interfaz de autodescripción de dispositivos		7*4 = 28			
Actividad 2: Desarrollo de una aplicación de teléfono móvil que implemente la interfaz anterior	2*4/2= 4		18*4/2 = 36 1*4 = 4	34*4 = 136	
Actividad 3: Desarrollo de dispositivos de pruebas					19*4/2 = 38

Estimación de horas por rol

Rol	Horas
TS	4
IS	28
AF	40
DE	136
DEMB	38
Total	** Expresión errónea **

Estimación horas/personal

Personal	Horas
Juan Gregorio Regalado Pacheco	246 horas (61.5 Días laborables a 4 horas al día)

Gestión de riesgos

Riesgo	Probabilidad	Acción correctora
Incapacidad para determinar parámetros generales de caracterización de dispositivos.	50,00%	Se limitará la funcionalidad de la aplicación para que pueda resolver el caso más general.
Dificultades para la generalización de la interfaz a otros sistemas embebidos.	20,00%	Se establecerá una plataforma embebida de referencia.
Problemas en los ciclos de testeo.	50,00%	Se dispone de un margen de tiempo considerable entre el 25 de mayo y el 12 de junio, fecha límite de entrega del proyecto.
Incidencias en la integración del soporte físico del dispositivo de prueba	20,00%	En la medida de lo posible se intentará recurrir a servicios de mecanizado profesional, lo cual debería minimizar esta probabilidad, en cualquier caso, se dispone de un margen de fechas considerable entre el 17 de noviembre de 2011 (fecha estimada de finalización del dispositivo de pruebas) y el 9 de enero de 2011.

Control de calidad

El punto crítico en el ámbito del control de calidad de este proyecto es, sin duda, el desarrollo de software y la integración del dispositivo de prueba:

- Como primera medida de gestión y aseguramiento de la calidad estableceremos, en una de las primeras tareas programadas, un sistema de control de versiones que nos permita llevar a cabo una regresión en el código en cualquier momento.
- Durante el desarrollo de la aplicación para dispositivos móviles se aplicará un enfoque de desarrollo basado en tests (Test driven development) que permita garantizar la consistencia e integridad entre las distintas versiones del software a medida que este evolucione.
- Debe preverse un plan de pruebas para el dispositivo físico, de modo que pueda constatarse su correcto funcionamiento en condiciones normales de operación.

Especificación de la interfaz Replicant.

Introducción.

Desde la década de los años ochenta del siglo pasado hemos asistido espectantes al desarrollo de la informática y a la penetración en los hogares de todo el mundo de ordenadores de sobremesa, portátiles, netbooks, teléfonos inteligentes y finalmente tablets. Sin embargo, lejos del foco mediático de atención, ha habido un sector que ha crecido aún más que el de la informática doméstica. Se trata de los sistemas embebidos.

Según Martín Cuenca, Angulo Usategui y otros[1],el mercado de microcontroladores en 1996 alcanzó la cifra de 2000 millones de unidades vendidas. Quince años después, con un ritmo de ventas en constante aumento, podríamos considerar los dispositivos embebidos basados en microcontroladores como la tecnología ubicua por excelencia.

Prácticamente cualquier equipo electrónico de consumo, sistema de comunicación, vehículo, sistema de control de acceso a edificios y una lista interminable de elementos similares, hace uso de un dispositivo microcontrolador. Además, con la aparición de la telefonía móvil y las redes inalámbricas, es cada vez más común encontrar sistemas que ofrecen la posibilidad de comunicarse con el exterior.

En este entorno, sin embargo, existe multitud de sistemas de comunicación distintos, dependientes del medio de transmisión y, en algunos casos, basados en protocolos de comunicación cerrados que dificultan el uso armónico de dispositivos que podrían complementarse entre sí para ofrecer la posibilidad de acometer tareas más complejas.

La interfaz que se propone en estas páginas es una primera aproximación a este problema de comunicación. La pretensión de este proyecto no es otra que establecer una posible vía de comunicación abierta, simple, independiente del medio de transmisión y extensible, que permita que distintos dispositivos que conviven en un entorno interconectado puedan expresar sus capacidades y ponerlas al servicio de otros con el fin de poder desarrollar tareas más complejas.

Este enfoque “bottom-up” ha servido en el ámbito de la tecnología para desarrollar éxitos sin precedentes que abarcan desde la simplicidad de la línea de comandos de las primeras estaciones de trabajo Unix hasta los actuales sistemas distribuidos basados en componentes.

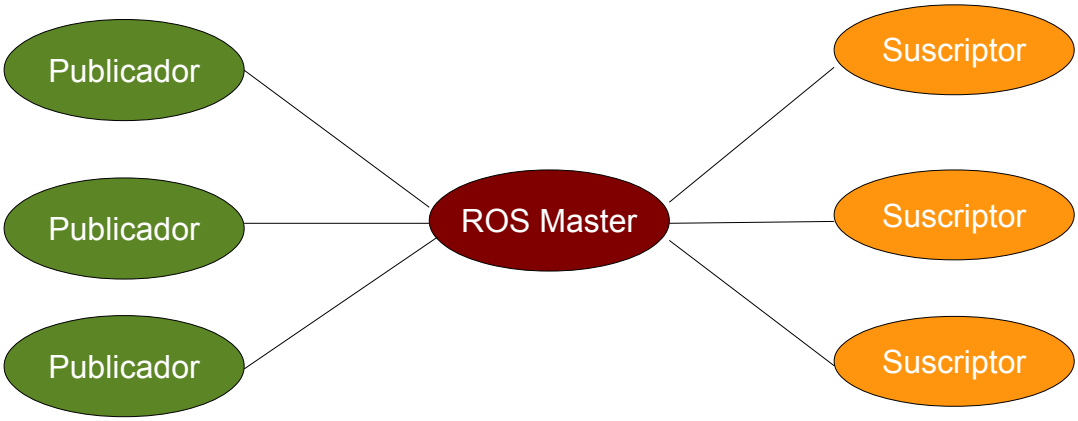
1 Microcontroladores PIC La clave del diseño. Autores: Eugenio Martín Cuenca, José Mª Angulo Usategui e Ignacio Angulo Martínez. Editorial Thomson Paraninfo.

Proyectos relacionados.

ROS.

ROS es el acrónimo de Robots Operating System. Es un proyecto basado en software libre y multiplataforma.

ROS está conformado por una pila de aplicaciones que implementan un patrón de publicación subscripción en el que distintos robots pueden mostrar las lecturas de sus sensores o enviar mensajes a un nodo central que los reenviará a todos los nodos que se hayan suscrito al canal concreto.



Dibujo 1: Arquitectura general ROS

Del mismo modo también se incluyen librerías de desarrollo para distintos lenguajes de programación entre los que se incluyen: Python, C++, Lisp, Java y Lua.

Es una plataforma que no está orientada al procesamiento en tiempo real, aunque puede lograrse esta característica en gran medida.

Por decisión de diseño, la capa implementada por ROS es realmente fina y poco intrusiva, de manera que la interfaz de comunicación habitual se basa en el paso de mensajes entre los distintos dispositivos y uno especial que actúa como servidor de nombres y que se conoce como el ROS Master.

Para tratar con las operaciones que requieren algún tipo de respuesta existe un nivel de comunicación adicional denominado Service. Un Nodo servidor ofrece el servicio y los nodos clientes lo invocan esperando por una respuesta.

ROS y Replicant presentan una serie de similitudes en relación con la tipología de dispositivos que parecen ser el objetivo de ambos proyectos, pero difieren en el hecho de que en el caso de Replicant la aplicación móvil no se concibe como un simple retransmisor de mensajes sino como un nodo de control centralizado.

En las conferencias Google I/O 2011 se planteó un escenario de uso de ROS en el ámbito de la robótica experimental y doméstica en el que se proponía la utilización de servicios “en la nube” para limitar los requisitos hardware y software de los robots, deslocalizando la ejecución de servicios que requieren una gran capacidad computacional y ejecutándolos como un servicio “de nube”

ADK.

A partir de la versión 2.3.4 de Android aparece una nueva interfaz para la interacción entre dispositivos USB y sistemas basados en Android.

Las únicas restricciones que debe cumplir un elemento para actuar como un accesorio USB para Android son implementar un protocolo simple conocido como Android accesory protocol y proveer 500 mA a 5V de alimentación entre sus conectores.

Este proyecto guarda relación directa con Replicant en el sentido de que abre la puerta a la interacción entre tablets, teléfonos inteligentes o pequeños netbooks y sistemas embebidos basados en microcontroladores a través de una interfaz USB.

La elección de Arduino como entorno de desarrollo para las primeras versiones de ADK no parece fortuita y puede venir dada por la facilidad de programación de dispositivos con prestaciones no triviales.

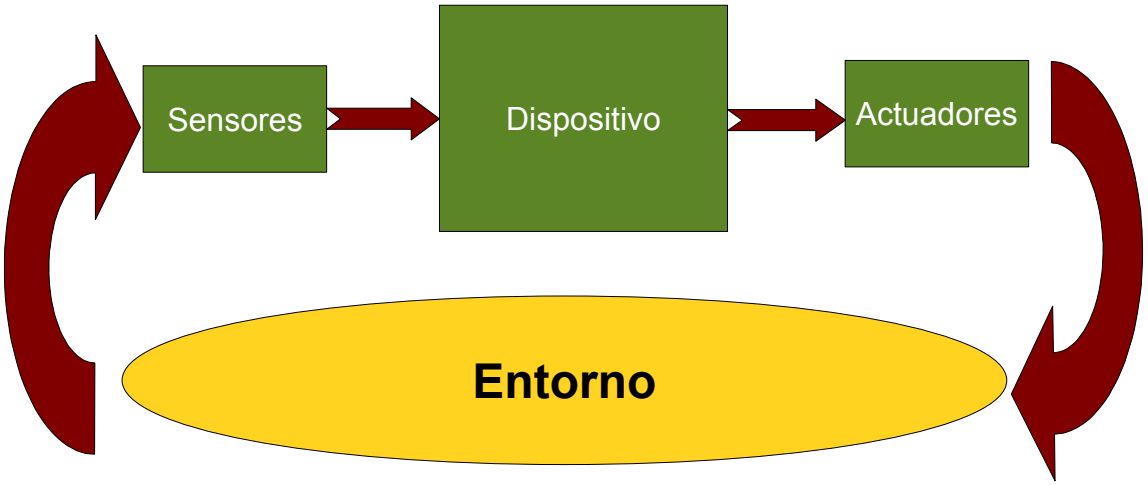
Arduino es un entorno basado en software libre para el desarrollo de sistemas embebidos, entre sus objetivos de diseño se encuentran ideas como la de valorar la sencillez en el desarrollo del software de control por encima de la eficiencia del código generado.

Arduino dispone de una comunidad de desarrolladores muy activa caracterizada por un gran espíritu de colaboración y un intenso impulso por derribar todas las barreras que suelen asociarse con el desarrollo de sistemas basados en microcontroladores.

El hecho de que las librerías para Arduino se desarrollen utilizando C++, un lenguaje orientado a objetos, lo hace un entorno interesante para una primera prueba de concepto de esta interfaz.

Caracterización de dispositivos.

Prácticamente toda la bibliografía de consulta habitual en los terrenos de la robótica y automática hace mención a la definición de los dispositivos embebidos como sistemas compuestos por elementos sensores que aportan información a una unidad de procesamiento que a su vez controla la acción de elementos actuadores que permiten hacer efectiva la interacción con el mundo físico.



Dibujo 2: Diagrama de bloques general de un robot o sistema automático.

Esta visión es bastante común pero poco precisa. Es necesario concretarla con un nivel de detalle más alto para poder abordar la definición de una interfaz adecuada para el control de este tipo de aparatos.

Caracterización de sensores.

En líneas generales, todos los sensores disponibles en el mercado podrían subdividirse en dos grandes grupos. Aquellos que devuelven sus lecturas como un valor continuo y aquellos que lo hacen digitalizándolas.

Caracterización de sensores en función de su tipo de salida:

- Sensores analógicos.
- Sensores digitales.

Para poder interpretar correctamente la información obtenida por los sensores, deberíamos conocer las unidades en las que se representan los resultados y la relación entre los datos percibidos y la salida, dado que no todos los sensores tienen una salida lineal con respecto al estímulo de entrada.

Existen también sensores que retornan valores simples como podría serla variación de la resistencia de una LDR y que podría representarse como un valor real positivo. O sensores que devuelven series de valores que representan información más compleja, como podría ser el caso de cámaras fotográficas, de video o micrófonos.

Otra posible clasificación de los sensores no guarda relación con el formato de su salida sino con su utilidad prevista. En función de este criterio planteamos una categorización de sensores que no pretende ser exhaustiva pero que puede servir para ilustrar la dificultad de intentar utilizar este enfoque para representarlos.

Caracterización de sensores en función de su utilidad prevista:

- Sensores de presencia (Todo/Nada)
  - Interruptores.
  - Finales de carrera.
- Sensores para la medición de distancias.
  - Sensores basados en ultrasonidos.
  - Sensores ópticos de reflexión.
- Sensores táctiles de presión.
  - Galgas extensiométricas.
  - Sensores resistivos de flexión.
- Sensores de presencia química.
  - Sensores de CO2
- Sensores de temperatura.
  - Termómetros.
  - Termopares.
- Sensores para la adquisición de imágenes.

    - Cámaras de fotos.
    - Cámaras de video.
- Sensores para la adquisición de sonidos.
    - Micrófonos.
- Sensores de posicionamiento y geolocalización.
    - Brújulas.
    - GPS's
    - Acelerómetros.
    - Giróscopos.

Las diferencias entre las dos categorizaciones de los sensores propuestas parecen más que evidentes. La simplicidad de la clasificación en función del tipo de salida también es innegable y hace que sea la opción seleccionada para la modelización dentro de esta interfaz.

Caracterización de actuadores.

Las posibles categorizaciones de los elementos actuadores presentan las mismas dificultades que las relacionadas con los sensores.

Un sistema embebido puede disponer de una gran variedad de actuadores, cada uno con una serie de características propias que definen sus capacidades finales.

Sin embargo, podríamos considerar que un actuador simplemente puede generar un giro en la estructura o un movimiento lineal. Los movimientos más complejos podrían modelarse a través de combinaciones de estas operaciones básicas.

También tendremos que considerar si se trata de un actuador con capacidad de retroalimentación o sin ella.

Tomando en consideración este enfoque podríamos plantear una clasificación de los diversos accionadores como la que se presenta a continuación.

- Actuadores con retroalimentación:
    - De revolución.
    - Lineales.
- Actuadores sin retroalimentación:
    - De revolución.
    - Lineales.
    - Diodos LED.
    - Altavoces
    - Pantallas

Con esta clasificación de actuadores y suficiente información sobre la disposición física de los mismos en la estructura del robot podríamos disponer de un modelo cinemático del mismo con un nivel de detalle aceptable. Además estaríamos en posesión de información suficiente para ofrecer retroalimentación visual o auditiva a otros dispositivos u operadores humanos.

Para poder modelar los aspectos dinámicos del dispositivo tendríamos que disponer de información relativa a la velocidad de giro o desplazamiento y su relación con los valores de entrada al dispositivo.

Definición de la interfaz Replicant.

Objetivos.

La interfaz Replicant debe permitir que cualquier dispositivo que la implemente describa sus elementos fundamentales, sean estos sensores o actuadores y aporte un cauce de comunicación que facilite el control del mismo, ya sea a través de la manipulación directa de los actuadores, o mediante la ejecución de órdenes más complejas.

La interfaz debe ser suficientemente simple como para poder desarrollarse sobre un sistema embebido cualquiera con capacidad de comunicación con el exterior.

Replicant no incluye ningún tipo de restricción sobre el medio de transmisión, aunque es cierto que su ámbito de aplicación natural se encuentra entre las conexiones de red inalámbricas.

Del mismo modo, esta interfaz no añade ninguna capa para la transmisión segura de la información, se entiende que el canal de comunicación entre los dispositivos replicant y los clientes replicant es seguro.

Criterios de diseño.

El principal criterio de diseño de esta interfaz es la simplicidad, se pretende que cualquier dispositivo basado en microcontroladores pueda implementar esta especificación, por lo tanto debe descartarse cualquier enfoque basado en la utilización de Remote Procedure Calls, Webservices o tecnologías similares que exceden con mucho la capacidad de cómputo de pequeños microcontroladores con recursos limitados.

Las consideraciones en cuanto a la seguridad del canal se dejan a cargo de los responsables del mismo, esta interfaz no introduce ninguna medida relacionada con la encriptación de los datos delegando esa responsabilidad a otras capas.

Dada la complejidad del desarrollo de sistemas basados en microcontroladores consideramos que es de vital importancia que el diseño de esta especificación tenga en cuenta, desde la etapa de diseño, la necesidad de poder trazar la ejecución de las distintas órdenes de manera amigable para el desarrollador.

Por último Replicant no introduce ninguna restricción en cuanto al medio de transmisión sobre el que se envían y reciben las distintas órdenes. Como criterio de diseño consideramos de vital importancia que esta interfaz pueda implementarse sobre cualquiera de las vías de comunicación habituales en el ámbito de los microcontroladores, abarcando desde las más básicas (comunicaciones serie a través de UART) hasta las más complejas (Comunicaciones a través de bluetooth, WiFi, GPRS, UMTS, EDGE, ZIGBEE, …)

Consideraciones sobre el uso de diagramas UML.

UML está pensado principalmente para el modelado de aplicaciones orientadas a objetos, sin embargo, en el entorno de los sistemas embebidos es habitual que el desarrollo sea procedimental. La elección de UML a la hora de representar los diagramas de esta interfaz se debe a dos factores fundamentales. Por un lado es un mecanismo de representación ampliamente utilizado en el sector y por otro, es claro y representa de una manera simple la interacción entre entidades.

Por lo tanto, el envío de mensajes entre entidades, en algunos casos puede ser representado mediante llamadas a métodos de objetos que no existirán como tales.

La implementación de esta interfaz, en el entorno de sistemas basados en microcontroladores se restringirá por lo tanto al reconocimiento de las diversas órdenes y a la respuesta adecuada a las mismas.

Arquitectura de la propuesta.

La interfaz Replicant establece dos grandes grupos de entidades, por un lado los dispositivos replicant (Denominados Replicant Devices en los diagramas de la especificación) y por otro lado las aplicaciones replicant (Denominados Replicant Applications en los diagramas)

Los dispositivos replicant se agrupan a través del concepto de comunidades, todos los dispositivos cuentan con un identificador único dentro de la comunidad a la que pertenecen, la comunidad, a su vez, se designa a través de una cadena de texto identificativa.

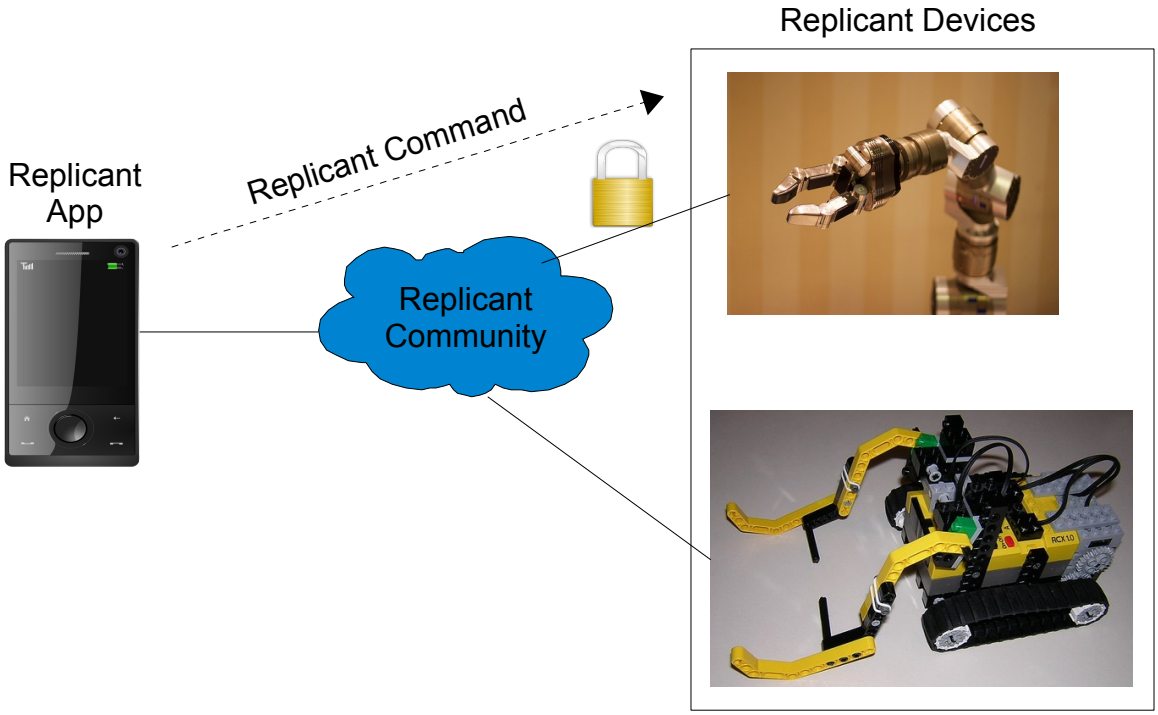
Un dispositivo replicant puede disponer de un modo autónomo de funcionamiento, pero se mantiene a la escucha de posibles peticiones por parte de aplicaciones replicant durante todo su ciclo de vida.

Los dispositivos replicant pueden recibir solicitudes simultáneas desde más de una aplicación replicant, pero en un instante determinado sólo pueden estar bloqueados por una única aplicación que podrá, a partir del momento en el que se confirme el bloqueo, enviar órdenes a los distintos actuadores del sistema embebido.

El bloqueo de un dispositivo está sujeto a un periodo de caducidad (en adelante timeout) que garantiza que el recurso esté siempre disponible para otras aplicaciones aún cuando una comunicación no haya sido cerrada de manera óptima.

Por último, una única aplicación replicant puede mantener bloqueados y por tanto controlar a la vez más de un dispositivo replicant.

El mecanismo de bloqueo es simple. Comienza cuando una aplicación solicita el bloqueo de un dispositivo, si el dispositivo no ha sido bloqueado previamente y la solicitud se realiza dentro de la comunidad a la que pertenece, el sistema embebido confirma el bloqueo



Proceso de inicialización de la comunicación.

El proceso de descubrimiento de los distintos dispositivos conectados no quedará cubierto por esta especificación, considerando legítimo cualquiera de los mecanismos habituales como pueden ser: Uso de paquetes ICMP, ping UDP, escaneo de puertos TCP, envío de cadenas de inicialización por una linea serie, bus I2C, ...

Una vez determinada la presencia de otros dispositivos comenzará la fase de reto o desafío en la que se intentará una conexión directa desde la aplicación cliente.

Tanto la aplicación como el dispositivo Replicant deben disponer de un parámetro configurado como tiempo de timeout, una vez expirado este tiempo sin haber recibido un nuevo dato, cualquiera de las dos partes implicadas en la comunicación pueden cerrar la conexión de manera unívoca.

Una vez establecida la conexión se enviará una cadena de texto identificativa de la comunidad de dispositivos representada mediante los códigos ASCII correspondientes y precedida por la cadena RepCom. Por lo tanto un mensaje tipo sería “RepCom <ComID>”

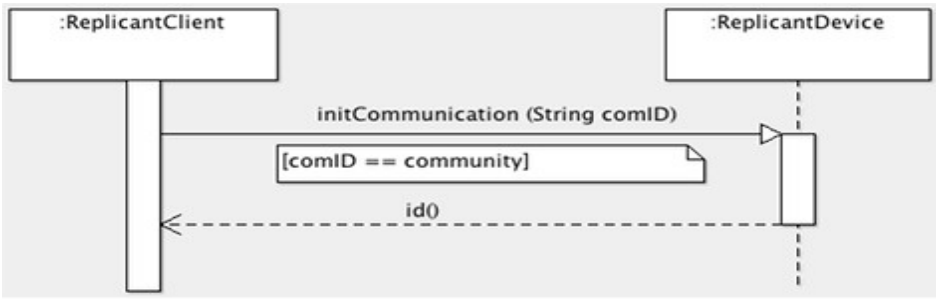


Ilustración 1: Proceso de inicialización de la comunicación.

El dispositivo comprobará si la cadena recibida se corresponde con su identificador de comunidad, en caso afirmativo debe responder con la siguiente cadena “RepDev <ID>” donde <ID> debe sustituirse por el identificador único, dentro de la comunidad, del dispositivo. Si la cadena de

identificación de comunidad no se correspondiese con la propia del dispositivo, éste debe ignorar el mensaje y continuar a la escucha.

Establecimiento de sesión y bloqueo del dispositivo.

Una vez garantizada la pertenencia a la misma comunidad, una aplicación cliente cualquiera puede solicitar el bloqueo de un dispositivo.

Para ello deberá enviar los siguientes datos “RepLock <CommunityID> <DevID> <AppID>” Dónde CommunityID representa el identificador de la comunidad, DevID el del dispositivo que se pretende bloquear y AppID el identificador de la aplicación cliente.

Ante este mensaje pueden darse dos situaciones. Que el dispositivo ya esté bloqueado por otra aplicación cliente, por lo tanto tendrá que rechazar la solicitud de bloqueo mediante un mensaje “RepLock Deny”. O que el dispositivo esté libre de bloqueo y por lo tanto acepte la petición mediante un mensaje “RepLock Accept”.

Si el dispositivo decide aceptar el bloqueo deberá almacenar el identificador de la aplicación que le bloquea para validar el resto de órdenes que reciba.

A partir de ese momento los dispositivos podrán intercambiar información entre ellos de manera fluída.

El bloqueo puede eliminarse en dos supuestos principales. El primero es que la aplicación u otro dispositivo decida cesar de utilizar al replicante bloqueado. El segundo es que transcurra un tiempo determinado sin actividad. En este caso será el propio replicante el que eliminará el bloqueo.

El tiempo de inactividad máximo se denominará timeout y será configurable. Cada vez que se recibe una nueva instrucción se reiniciará la cuenta de timeout y, por lo tanto, se ampliará el tiempo de bloqueo.

Ilustración 2: Proceso de bloqueo de un dispositivo.

La secuencia de desbloqueo será similar a la anterior. El dispositivo que solicita el desbloqueo enviará el mensaje “RepUnlock <CommunityID> <DevID> <AppID>”.

El dispositivo sólo liberará el bloqueo si ha sido solicitado por la aplicación que lo mantenía bloqueado.

Opcionalmente, el dispositivo que recibe la solicitud de desbloqueo puede responder con un mensaje “RepUnlock Accept”

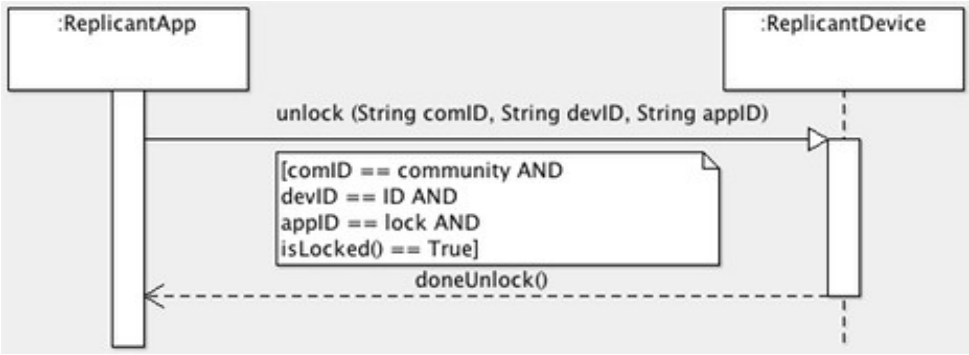


Ilustración 3: Proceso de desbloqueo de un dispositivo.

Proceso de consulta de sensores y actuadores.

Las únicas operaciones que no requieren de un bloqueo previo del dispositivo son las de consulta de sensores y actuadores. Estas operaciones sirven para que cualquier aplicación replicant pueda explorar a cualquier dispositivo de una comunidad dada y conocer sus capacidades.

En ambas consultas la aplicación cliente debe enviar el identificador de comunidad y el del dispositivo del que quiere obtener información. Si el dispositivo replicant es capaz de verificar que el mensaje va dirigido a la comunidad a la que pertenece y que contiene su identificador entonces responderá a la solicitud.

La solicitud de información sobre los sensores seguirá la siguiente secuencia:

En primer lugar la aplicación cliente enviará un mensaje con los siguientes datos "RepSensorsQuery <CommunityID> <DevID>"

El dispositivo responderá mediante un mensaje con el siguiente contenido:

"RepSensorsResponse <ID> <SensorCount> <Sensor1ID> <Sensor1Name> <Sensor1Units> <Sensor1IsDigital> … <SensornID> <SensorNName> <SensornUnits> <SensornIsDigital>"

Donde <ID> representa el identificador unívoco dentro de la comunidad del dispositivo replicant. <SensorCount> representa el número total de sensores que se están enviando.

<SensorxID> representa el identificador del sensor x dentro del dispositivo.

<SensorxName> representa una cadena de texto identificativa del sensor x dentro del dispositivo.

<SensorxUnits> representa las unidades en las que se devuelve el valor medido por el sensor una vez procesado.

<SensorxIsDigital> representa, cuando tiene un valor "true" que la salida del sensor es digital. Cuando tiene un valor "false" representa que la salida del sensor es analógica.

En el caso de la consulta de actuadores el proceso es similar al anterior. El cliente envía la siguiente cadena a través de los medios de transmisión disponibles. "RepActuatorsQuery <CommunityID> <DevID>" los argumentos de esta cadena conservan el sentido semántico de los de la anterior.

La respuesta difiere solamente en los parámetros de caracterización de los actuadores, quedando como sigue.

"RepActuatorsResponse <ID> <ActuatorCount> <Actuator1ID> <Actuator1Name> <Actuator1MinValue> <Actuator1MaxValue> <Actuator1HasFeedback> … <ActuatornID> <ActuatornName> <ActuatornMinValue> <ActuatornMaxValue> <ActuatornHasFeedback>"

Dónde, una vez más <ID> corresponde al identificador unívoco del dispositivo dentro de la comunidad.

<ActuatorCount> Indica la cantidad de actuadores presentes en el dispositivo.

<ActuatorxID> Representa el identificador correspondiente al actuador x dentro del dispositivo.

<ActuatorxName> Representa una cadena de texto identificativa del actuador x dentro del dispositivo.

<ActuatorxMinValue> Representa el valor mínimo que puede recibir como parámetro de operación el actuador x.

<ActuatorxMaxValue> Representa el valor máximo que puede recibir como parámetro de operación el actuador x.

<ActuatorxHasFeedback> Representa, cuando toma el valor "true", que el actuador devuelve un valor de realimentación cuando recibe una orden. Cuando toma el valor "false" indica que se trata de un actuador en bucle abierto y no proporciona información sobre el resultado de una orden.

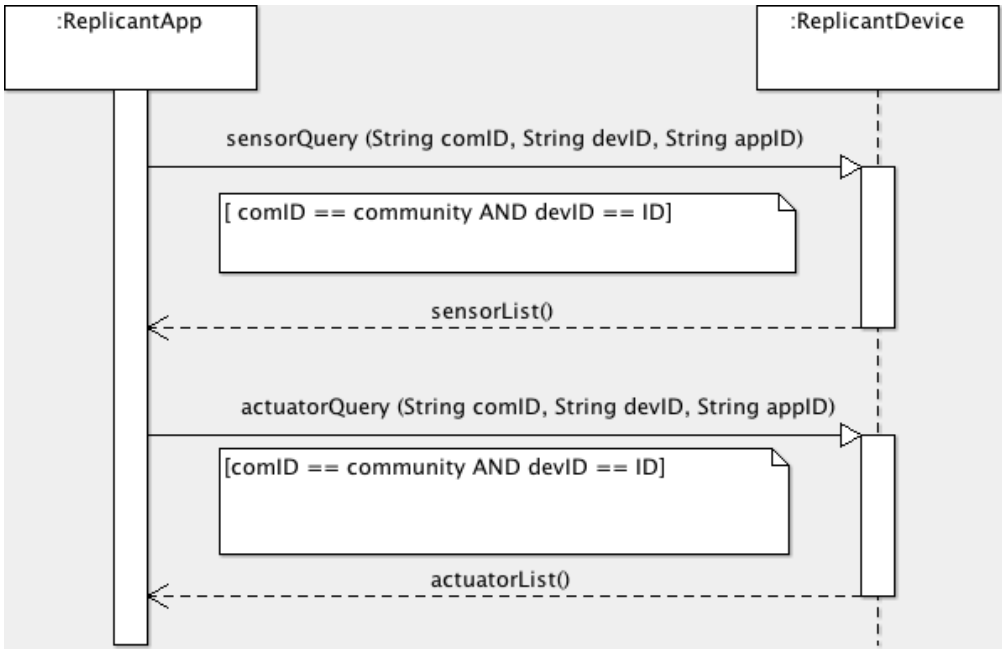


Ilustración 4: Proceso de consulta de sensores y actuadores.

Proceso de control remoto del dispositivo.

En cuanto la aplicación cliente ha bloqueado un dispositivo y dispone de información sobre los módulos que lo componen, puede comenzar a interactuar con él.

La interacción con los sensores se basará en la recepción de los valores de lectura de los mismos, pudiendo tratarse de valores de lectura directa, es decir, lo que el sensor transmite directamente a su salida, o valores de lectura procesada, es decir, los valores que se han obtenido convertidos a las unidades de salida tras haber sido interpretados según las especificaciones del fabricante. Habitualmente, la recepción de las lecturas de los sensores servirá para llevar a cabo una representación gráfica de las mismas.

El proceso de obtención de los datos de un sensor es el siguiente. La aplicación cliente envía un mensaje con la cadena “RepGetRawSensor <CommunityID> <DevID> <AppID> <SensorID>” en el caso de solicitar una lectura literal o “RepGetProcSensor <CommunityID> <DevId> <AppID> <SensorID>” en el caso de una lectura procesada.

Dónde <AppID> se corresponde con el identificador de la aplicación cliente.

<CommunityID> se refiere al identificador de la comunidad.

<DevID> se refiere al dispositivo al que está conectado el sensor.

<SensorID> se refiere al identificador del sensor del dispositivo al que está conectado.

La respuesta a este mensaje debe ser otro con el siguiente contenido “RepRawSensorVal <ID> <SensorID> <RawValue>” en el caso de una solicitud de valor literal o “RepProcSensorVal <ID> <SensorID> <ProcessedValue>” en el caso de una solicitud de valor procesado.

En ambos mensajes <ID> se refiere al identificador del dispositivo.

<SensorID> se refiere al identificador del sensor.

<RawValue> Se refiere a un valor literal obtenido directamente de la salida del sensor.

<ProcessedValue> Se refiere a un valor procesado por el propio sistema embebido después de su lectura desde la salida del sensor.

En cualquiera de los casos anteriores, el valor devuelto debe ser simple, representado por una cadena, un entero o un flotante.

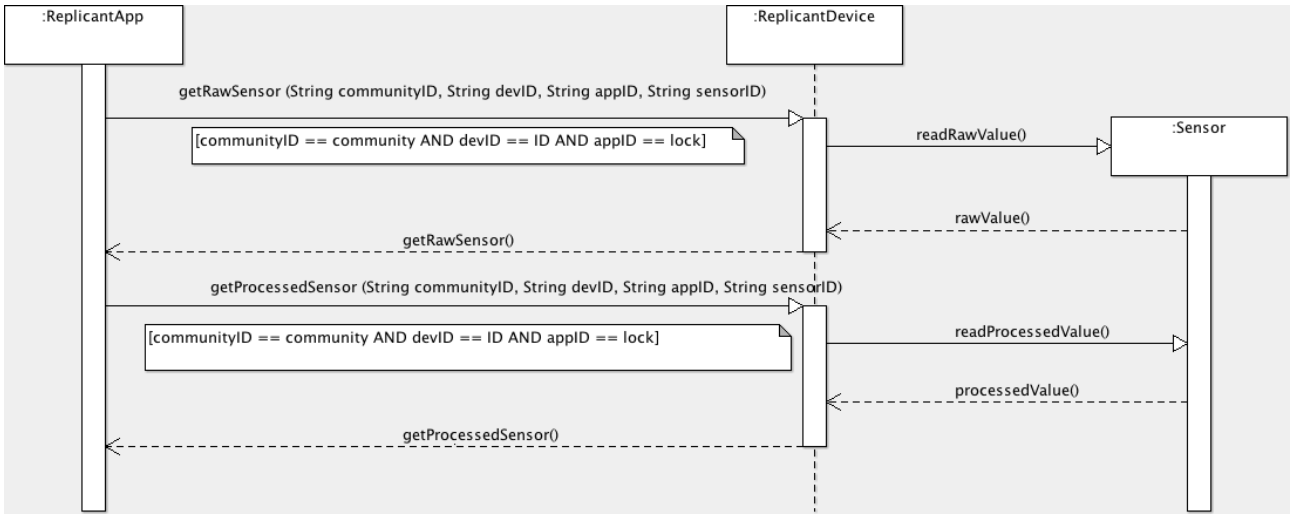


Ilustración 5: Proceso de interacción con sensores.

Por último, la manera de enviar órdenes a los actuadores se basa en el uso de dos mensajes. Uno para la interacción con actuadores convencionales y otro relacionado con aquellos actuadores con capacidad de retroalimentación.

En cualquier caso, la estructura básica del mensaje es similar.

Para el control de un actuador sin realimentación debe utilizarse “RepSetActuator <CommunityID> <DevID> <AppID> <ActuatorID> <Value>” este mensaje no recibe ninguna respuesta por parte del dispositivo.

Para el control de una actuador con realimentación se utilizará “RepSetFeedbackActuator <CommunityID> <DevID> <AppID> <ActuatorID> <Value>”

En este caso <AppID> representa siempre el identificador de la aplicación cliente.

<CommunityID> se refiere a la cadena de identificación de la comunidad.

<DevID> sirve para identificar el dispositivo.

<ActuatorID> hace referencia al identificador del actuador.

<Value> es el valor que quiere transmitirse para el actuador concreto.

En respuesta a una orden RepSetFeedbackActuator se recibirá un mensaje con el siguiente formato: "RepFeedbackActuatorVal <ID> <ActuatorID> <Value>"

<ID> Representa la identificación del dispositivo que responde.

<ActuatorID> Representa el identificador del actuador.

<Value> Representa el valor efectivo que ha llegado a adquirir dicho actuador.

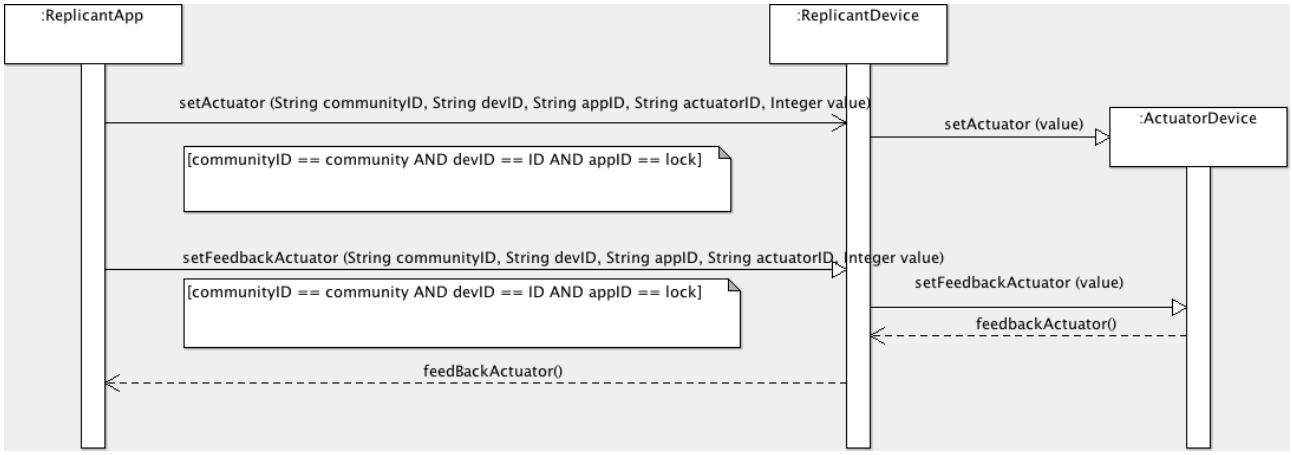
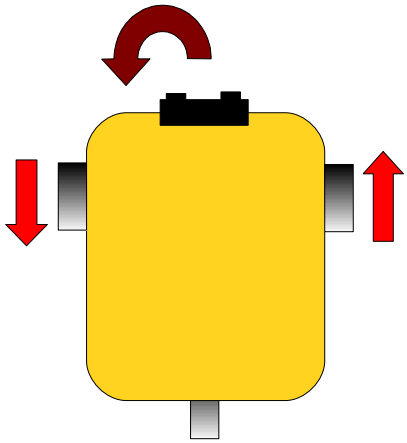


Ilustración 6: Proceso de interacción con actuadores.

Descripción de operaciones compuestas.

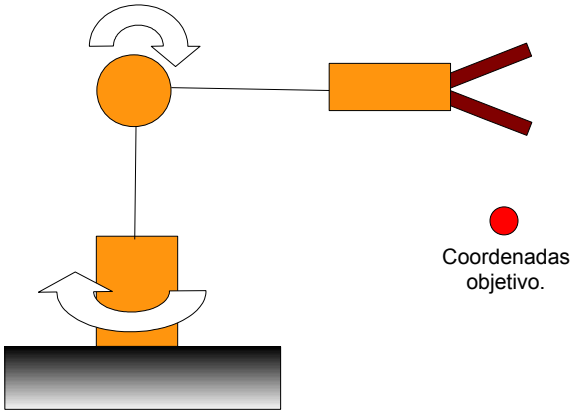
El programador de un dispositivo replicant puede definir operaciones compuestas de más alto nivel.

Por ejemplo, un robot móvil con dos motores y sistema de dirección diferencial, para realizar un giro cerrado a la izquierda debería llevar a cabo una operación en la que el motor izquierdo gira hacia detrás y el motor derecho gira hacia delante a la misma velocidad.



Dibujo 3: Representación de una operación compuesta sobre un robot móvil

En el caso de un manipulador robótico, la operación de establecer la posición de la pinza en unas coordenadas espaciales determinadas se traducirá en el giro de prácticamente todas sus articulaciones.



Dibujo 4: Representación de una operación compuesta sobre un manipulador.

A pesar de que los dos ejemplos anteriores difieren profundamente en cuanto a la complejidad y a las acciones a acometer para conseguir los diversos objetivos tienen, sin embargo, un aspecto en común. En ningún caso podríamos saber de antemano todas las operaciones compuestas que un desarrollador va a implementar para cada uno de sus sistemas.

La posibilidad de definir estas características para un dispositivo replicant añadiría un gran valor a esta interfaz y permitiría la definición de rutinas de trabajo productivas entre distintos dispositivos.

Una aplicación replicant puede consultar a un dispositivo replicant por las operaciones compuestas que puede llevar a cabo. En respuesta a esta consulta, el dispositivo replicant devolverá una lista de identificadores de operaciones.

Una vez obtenida la lista de identificadores de operaciones, la aplicación replicant puede solicitar más detalles de algunas de estas operaciones con lo que el dispositivo replicant le enviará una descripción textual sobre los efectos del uso de dicha operación y el número y tipo de argumentos que espera.

En cuanto se dispone de toda esta información, la aplicación podrá enviar las órdenes precisas con los argumentos necesarios para utilizar estas funciones complejas.

La solicitud de la lista de operaciones se lleva a cabo mediante la cadena “RepCompOp <CommunityID> <DevID> <AppID>”

Los parámetros tienen el mismo significado que en el resto de operaciones.

En respuesta a esta consulta el dispositivo replicant enviaría una nueva cadena con el siguiente formato:

“RepComOpResponse <ID> <NumOps> <OP1ID> <OP2ID> … <OpnID>”

<ID> representa el identificador unívoco dentro de la comunidad.

<NumOps> Representa el número de operaciones compuestas que se han definido sobre el dispositivo replicant en cuestión.

<OpXID> Representa el identificador corto de la operación compuesta X.

Una vez se dispone de la lista de operaciones compuestas la aplicación cliente puede solicitar más información sobre una operación determinada. Para ello se dispone del mensaje:

“RepComOpDetail <CommunityID> <DevID> <AppID> <OPID>”

Los tres primeros parámetros son los habituales en el resto de órdenes.

<OPID> Representa el identificador de la orden de la que se espera obtener información más detallada.

En respuesta a un mensaje de este tipo el dispositivo replicant enviará un mensaje con el siguiente formato.

“RepComOpDetailResp <ID> <OPID> <OPName> <NumArgs> <Arg1ID> <Arg1Name> <Arg1Description> <Arg2ID> <Arg2Name> <Arg2Description> … <ArgNID> <ArgNName> <ArgNDescription> <OPDetailedDescription>”

Dónde <ID> es el identificador unívoco del dispositivo replicant dentro de su comunidad.

<OPID> Representa el identificador de la operación compuesta.

<OpName> Representa un texto corto que da nombre a la operación.

<NumArgs> Representa el número de argumentos totales que acepta la operación.

<ArgXID> Representa el tipo de dato del argumento X.

<ArgXName> Representa el nombre del argumento.

<ArgXDescription> Representa a un pequeño texto descriptivo sobre los posibles valores que puede tomar el argumento X.

<OPDetailedDescription> Representa un texto largo en el que se puede describir con precisión cuál es la función de la operación compuesta en cuestión.

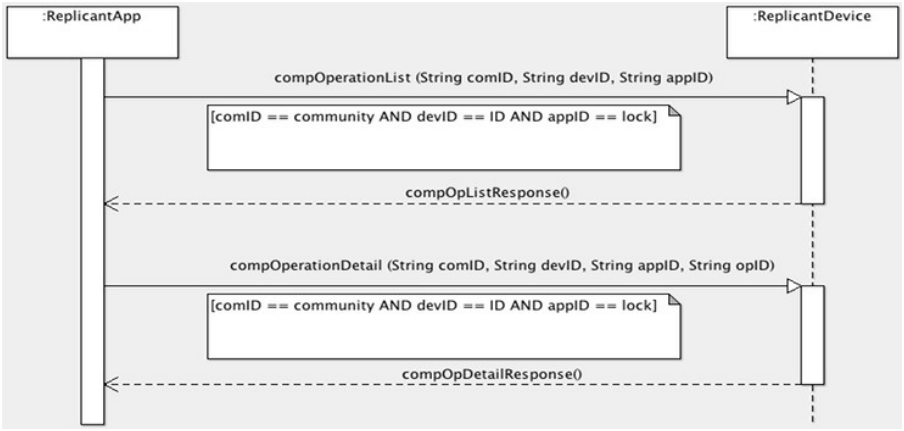


Ilustración 7: Proceso de consulta de operaciones compuestas.

Para ejecutar una operación compuesta se enviará la siguiente orden “RepComOpRun <CommunityID> <DevID> <AppID> <OPID> <Arg1Val> <Arg2Val> … <ArgnVal>”

Dónde los tres primeros argumentos tienen el mismo sentido que en el resto de mensajes.

<OPID> Representa el identificador de la operación compuesta.

<ArgXVal> Representa el valor que se pasa como argumento X.

Es responsabilidad de la aplicación replicant mantener la correspondencia entre órdenes compuestas y número de argumentos así como la de controlar los tipos de datos de los mismos.

Las órdenes de ejecución de operaciones compuestas no tienen respuesta aún cuando afecten a

actuadores con retroalimentación.

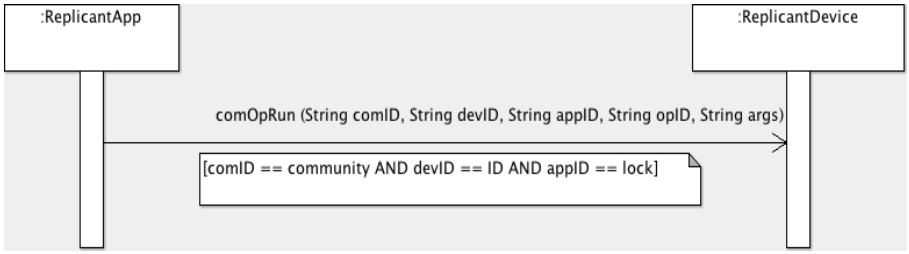


Ilustración 8: Proceso de ejecución de operaciones complejas.

Suscripción a eventos.

Durante el ciclo de vida de cualquier dispositivo embebido, existen numerosas ocasiones en las que pueden ocurrir eventos que surgen de manera asíncrona e imprevisible.

Habitualmente estos eventos se tratan a partir del uso de interrupciones. La aparición de una interrupción habitualmente para la ejecución del bucle principal del programa de control del dispositivo y provoca la bifurcación del flujo de instrucciones hacia procedimientos que suelen denominarse rutinas de atención a interrupciones, estos procedimientos deben ser sencillos y eficientes dado que durante el tiempo de atención a interrupciones el dispositivo no puede continuar atendiendo a su operación normal.

Muchas interrupciones provocan actualizaciones de contadores internos y temporizadores, pero otras pueden alertar al dispositivo de situaciones que ponen en riesgo su propia integridad física.

Parece claro que esta especificación no estaría completa sin un mecanismo que habilitase a los clientes replicant para recibir notificaciones ante el desencadenamiento de uno de estos eventos.

Dado que la casuística de los posibles eventos desencadenantes de una interrupción es prácticamente inabarcable, parece lógico proponer un sistema de publicación suscripción similar al auspiciado por el framework ROS.

El diseñador del dispositivo será el responsable de registrar aquellos eventos que considera que pueden ser del interés del resto de la comunidad.

La aplicación replicant podrá consultar el listado de eventos disponibles y solicitar la suscripción a las notificaciones de aquellos que considere oportuno.

El proceso de consulta consistirá en el envío del siguiente mensaje:

“RepEventsQuery <CommunityID> <DevID> <AppID>”

Como es usual en el resto de mensajes, el primer argumento corresponderá a la comunidad a la que pertenece el dispositivo, el segundo será el identificador unívoco del dispositivo dentro de su comunidad y el último será el identificador de la aplicación replicant.

Antes de responder el dispositivo deberá comprobar que el identificador de la aplicación corresponde con el de la que lo ha bloqueado.

En respuesta a este mensaje, el dispositivo enviará una lista con los eventos que publica.

“RepEventsResponse <ID> <EventsCount> <Event1ID> <Event1Name> <Event1Units> <Event1Doc> … <EventnID> <EventnName> <EventnUnits> <EventnDoc>”

Los argumentos de este mensaje representan los siguientes datos:

<ID> Identificador unívoco del dispositivo dentro de su comunidad.

<EventsCount> Número de eventos registrados en el dispositivo a los que la aplicación cliente puede suscribirse.

<EventXID> Identificador del evento X.

<EventXName> Nombre del evento X.

<EventXUnits> Unidades en las que se expresa el valor devuelto por el evento X.

<EventXDoc> Pequeña cadena descriptiva sobre el evento X.

Una vez obtenida la lista de eventos publicados por el dispositivo, la aplicación replicant podrá suscribirse a cualquiera de estos eventos para recibir notificaciones cuando se produzca cualquier incidencia. Para ello solamente tendrá que enviar el siguiente mensaje.

“RepEventSuscribe <CommunityID> <DevID> <AppID> <EventID>”

Los tres primeros argumentos se corresponden con los identificadores de comunidad, dispositivo y aplicación respectivamente y el último argumento, <EventID> representa el evento al que se pretende suscribir.

Finalmente, el dispositivo responderá con un mensaje:

“RepEventSuscribeOK <ID> <EventID>”

En el que se indica que se ha suscrito correctamente al evento <EventID> del dispositivo <ID>.

Los mensajes que se envían una vez se ha establecido la suscripción mantendrán siempre la siguiente estructura sintáctica.

“RepEventMsg <ID> <EventID> <Value>”

Donde los argumentos representan los siguientes datos:

<ID> Identificador unívoco del dispositivo dentro de la comunidad.

<EventID> Identificador del evento dentro del dispositivo.

<Value> Valor generado por la aparición del evento.

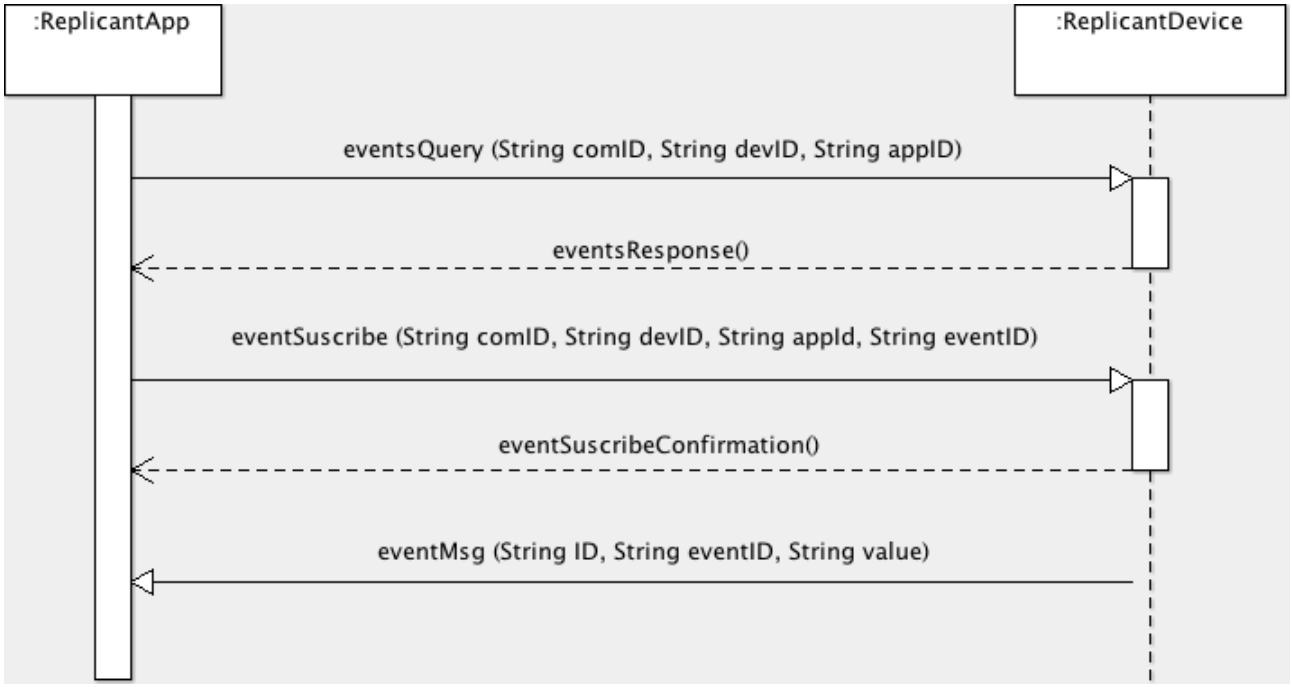


Ilustración 9: Proceso de consulta y suscripción a eventos

Finalización de la comunicación.

La comunicación puede finalizarse por parte de la aplicación replicant o por parte del dispositivo replicant. En cualquier caso, el formato del mensaje es "RepComClose <CommunityID> <DevID> <AppID>"

Este mensaje no recibe ninguna respuesta.

Desarrollo del dispositivo de pruebas.

Objetivos de diseño.

Replicant es un sistema para la intercomunicación de dispositivos. Parte de la inspiración de esta interfaz proviene del mundo de la robótica. Éste es el principal motivo de plantear como parte integrante del proyecto el desarrollo de un pequeño robot móvil que implemente toda la funcionalidad requerida para soportar la especificación Replicant.

Dado que se trata del primer replicante, se ha denominado al robot como Rachel.

Los objetivos principales de diseño de este dispositivo son los siguientes:

- Mostrar la aplicabilidad directa de Replicant como plataforma para la interacción de máquinas con capacidad de comunicación.
- Desarrollar un dispositivo sencillo y fácilmente reproducible que permita comprobar el funcionamiento de las distintas características de la interfaz.
- Proponer un diseño de bajo costo que pueda utilizarse como plataforma educativa en el futuro.
- Estudiar la viabilidad de la implantación de Replicant en plataformas de desarrollo basadas

en microcontroladores con recursos limitados.

Elección de la plataforma de desarrollo.

A la hora de seleccionar la plataforma para el desarrollo de este proyecto se han tomado en consideración dos factores principales:

El primer factor a tener en cuenta ha sido la consideración sobre la dificultad para comenzar a desarrollar sobre una plataforma dada.

En el mundo de las aplicaciones embebidas es habitual disponer de herramientas realmente espartanas y que no están enfocadas a la productividad durante el proceso de desarrollo. Esto unido a que suelen utilizarse lenguajes de muy bajo nivel para garantizar la optimización del resultado final, dificulta en gran medida la posibilidad de implementar un sistema complejo en un plazo tan corto como el que se ha fijado para la finalización de este proyecto.

Por lo tanto se han primado las plataformas que permiten el desarrollo de sistemas mediante el uso de lenguajes de alto nivel y aquellas cuyos entornos de desarrollo fueran gratuitos y con una documentación suficientemente clara y accesible.

El segundo factor determinante a la hora de seleccionar la plataforma se ha enfocado a la posibilidad de generalizar el resultado sobre una gama suficientemente amplia de dispositivos.

El mercado actual incluye desde microcontroladores de 8 bits con recursos escasos hasta sistemas de 32 bits con interfaces de red inalámbrica integradas.

Para el desarrollo del dispositivo de pruebas hemos optado por la utilización del sistema más austero posible, de modo que los resultados obtenidos sobre esta plataforma pudieran extrapolarse a cualquier otra.

Las opciones valoradas han sido:

	Mbed NXP LPC1768	Arduino UNO	PIC 16F870
Microcontrolador	ARM-Cortex M3	ATMEL Atmega AVR328P	MicrochipPIC 16F870
Frecuencia de trabajo	96 MHz	16 MHz	20MHz
Flash	512Kb	32Kb	2K
Ram	32Kb	2Kb	128B
Ethernet	X		
WiFI			
UART	X	X	X
SPI	X	X	
I2C	X	X	
USB	X	X	
Entradas analógicas	6	6	5
Salidas analógicas	6	6	
Entradas/Salidas digitales	25	14	22

Lenguaje de programación	C, C++	Similar a C, C++	Ensamblador, C.
IDE	Basado en Web	Arduino IDE	MPLAB
IDE Gratuito	SÍ	SÍ	SÍ
IDE Multiplataforma	SÍ	SÍ	No

Finalmente la opción seleccionada ha sido la plataforma Arduino dado que cumple con todos los requisitos preestablecidos.

No utiliza un microcontrolador de gama alta, sus características en cuanto a disponibilidad de memoria flash y RAM le sitúan como una buena opción a la hora de garantizar la generalización de los resultados y dispone de gran cantidad de interfaces de comunicación con el exterior.

Por otro lado su entorno de desarrollo está basado en software libre y es multiplataforma, dispone de resaltado de sintaxis y además es sencillo de utilizar.

Como complemento a la plataforma de desarrollo seleccionada se ha adquirido una interfaz 802.11 b/g denominada WiFLY Shield y basada en la utilización del módulo WiFly RN-131C de la compañía Roving Networks.

Definición de módulos funcionales.

A nivel funcional podemos considerar que el robot móvil está formado por los siguientes bloques funcionales:

El primero y más importante es la unidad de proceso principal, formada por la propia placa Arduino UNO a la que se conectan el resto de dispositivos.

El siguiente bloque, clasificado por nivel de complejidad, sería la interfaz de red inalámbrica WiFly de Roving Networks. Provee al sistema con conectividad 802.11 b/g y soporta los sistemas de encriptación WEP, WPA y WPA2-PSK. Dispone de una antena cerámica integrada y conector para una antena exterior si fuera necesaria. Permite llevar a cabo transferencias de, como máximo, 4Mbps.

Se conecta a Arduino a través de una interfaz SPI y se presenta en formato shield, denominación que se utiliza en la comunidad de desarrolladores de esta plataforma para designar a las placas diseñadas para colocarse encima de un Arduino original.

A continuación disponemos del primer sensor, un dispositivo SHARP GP2Y0A02YK0F. Se trata, básicamente, de un sensor para la medición de distancias basado en la utilización de infrarrojos y capaz de detectar la presencia de objetos que se encuentran en el rango que va de los 20 a los 150 cm.

La salida de este sensor es analógica y sigue la siguiente curva según las especificaciones del fabricante:

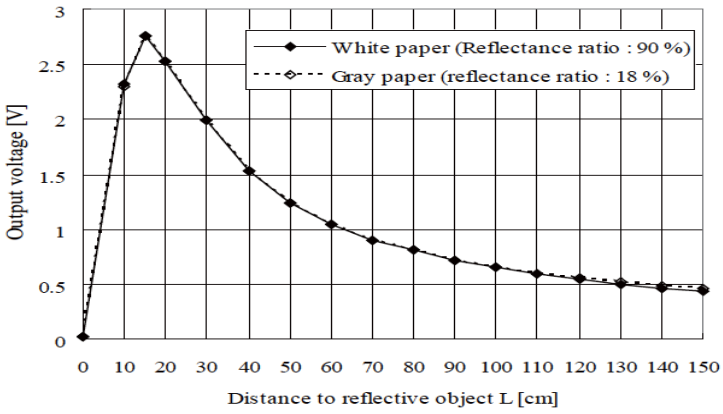


Ilustración 10: Curva de salida del sensor SHARP.

El robot móvil monta en su frontal un medidor ultrasónico de distancias MaxSonar-EZ3 de la compañía Maxbotics. La salida de este elemento es también analógica pero lineal, devuelve 3,86 mV por centímetro a la salida y permite determinar la distancia de objetos situados hasta 6,45 m.

Para poder mostrar el funcionamiento de algún actuador con retroalimentación (elemento contemplado en el desarrollo de la especificación Replicant) se han añadido dos sensores ópticos de reflexión QRD1114 de la compañía Fairchild. Estos sensores se colocan en el interior de las ruedas, apuntando directamente a un encoder óptico diseñado específicamente para este robot. Permiten estimar cuánto ha girado una rueda del mismo.

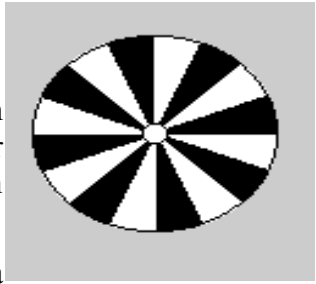


Ilustración 11: Encoder óptico.

La intención original era instalar dos sensores por rueda y diseñar un encoder con dos franjas desfasadas. Este diseño permitía, no sólo saber cuánto se ha desplazado una rueda sino también en qué sentido. Sin embargo, una limitación del propio Arduino frustró esta iniciativa.

Arduino UNO sólo dispone de dos entradas de interrupción externa. La monitorización de los cambios de un encoder es totalmente inviable desde un enfoque basado en polling[2], por lo tanto, el diseño original requeriría de cuatro interrupciones.

Para finalizar el capítulo de sensores, el robot móvil dispone de dos resistencias LDR, también conocidas como fotocélulas. Se trata de unos componentes electrónicos discretos cuya resistencia varía en función de la luminosidad presente. Se conectan a la placa Arduino a través de sendas entradas analógicas y permiten estimar la cantidad de luz presente en la sala por la que se desplaza el dispositivo.

En el ámbito de actuadores se dispone de dos diodos LED frontales que pueden apagarse o encenderse independientemente y de dos servos de rotación continua.

Los servos son motores que disponen de una caja reductora de engranajes que transforma la alta velocidad de giro en el eje de salida de cualquier motor de corriente continua en un par de giro mayor. Suelen funcionar mediante una entrada de control que recibe una señal codificada mediante modulación por anchura de pulsos (PWM) y que representa el giro que el motor debe llevar a cabo.

2 Polling: Se denomina polling a un sistema de monitorización que consiste en ciclos cerrados en los que, por turnos, se va consultando el estado de una entrada. Se ha usado el término inglés por ser el más habitual en la bibliografía.

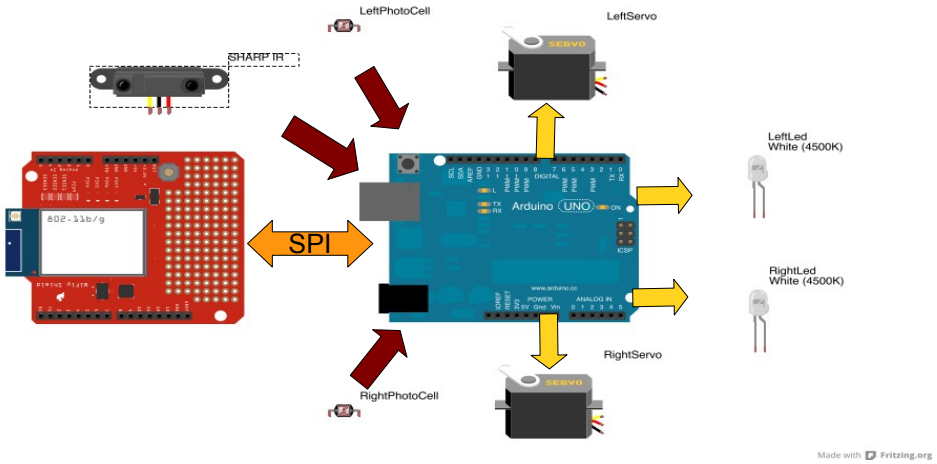


Ilustración 12: Estructura de bloques.

Diseño del soporte físico.

Para garantizar la facilidad a la hora de reproducir el diseño del robot se ha optado por utilizar materiales estándar.

El cuerpo principal de Rachel se ha construido a partir de un perfil de aluminio en L de 2 mm de grosor y 25x25 mm.

Los soportes para los servos se han construido a partir de un perfil de aluminio en U de 2 mm de grosor y 20x20 mm.

El soporte para el eje trasero se ha diseñado utilizando como base una varilla plana de 2 mm de grosor y 25 mm. de ancho.

Los soportes de las ruedas frontales se unen al cuerpo principal mediante escuadras de 50x50 mm.

El eje vertical de la rueda trasera se ha fabricado a partir de un trozo de varilla cilíndrica de 6 mm de diámetro y 45 mm de alto.

Finalmente, sobre el cuerpo principal se deposita una lámina de PVC multicell de 4 mm. de grosor.

Todos los taladros realizados en el cuerpo tienen un diámetro de 3 mm con la excepción de los correspondientes al alojamiento de los dos diodos LED frontales y del eje vertical de la rueda trasera, estos últimos tienen un diámetro de 5 mm.

Vistas frontal y posterior del cuerpo principal del dispositivo.

En los laterales del frontal se encuentran dos taladros de 3 mm. De diámetro que sirven como alojamiento de los dos diodos LED.

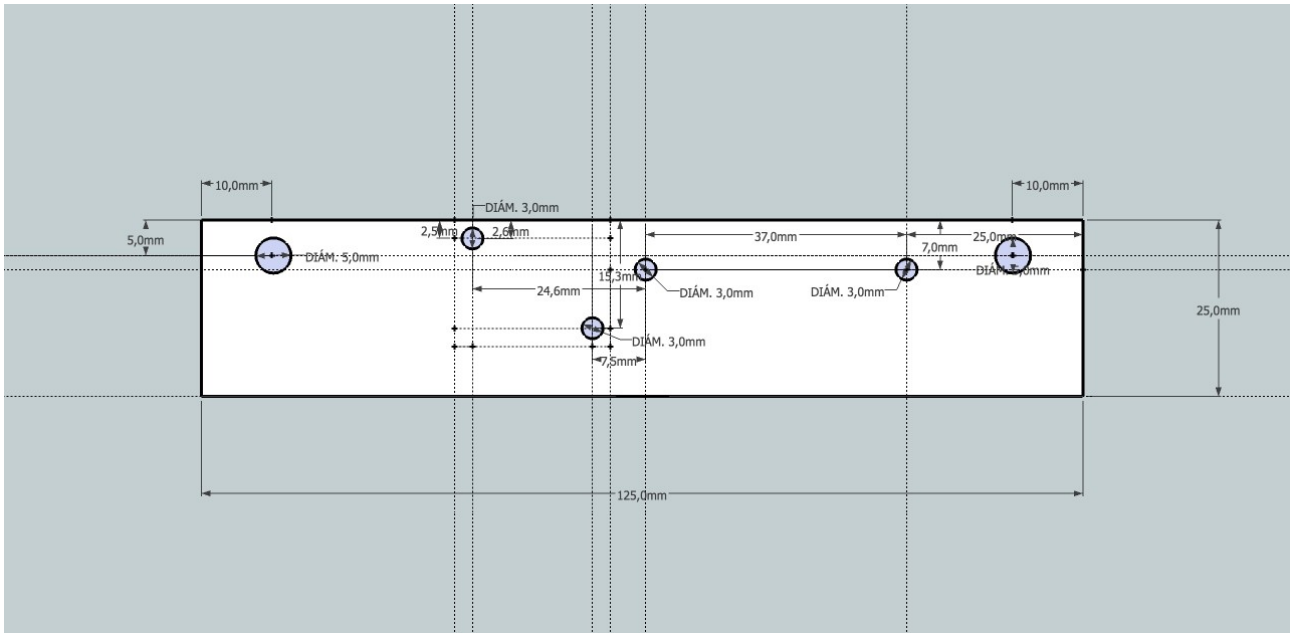


Ilustración 13: Vista frontal del cuerpo principal del dispositivo.

En el lateral izquierdo se encuentran dos taladros en diagonal que se utilizan para fijar en su posición definitiva el módulo de ultrasonidos e inmediatamente a su derecha se encuentran otros dos orificios destinados a sujetar el sensor de infrarrojos.

La parte posterior del robot no aloja ningún componente específico.

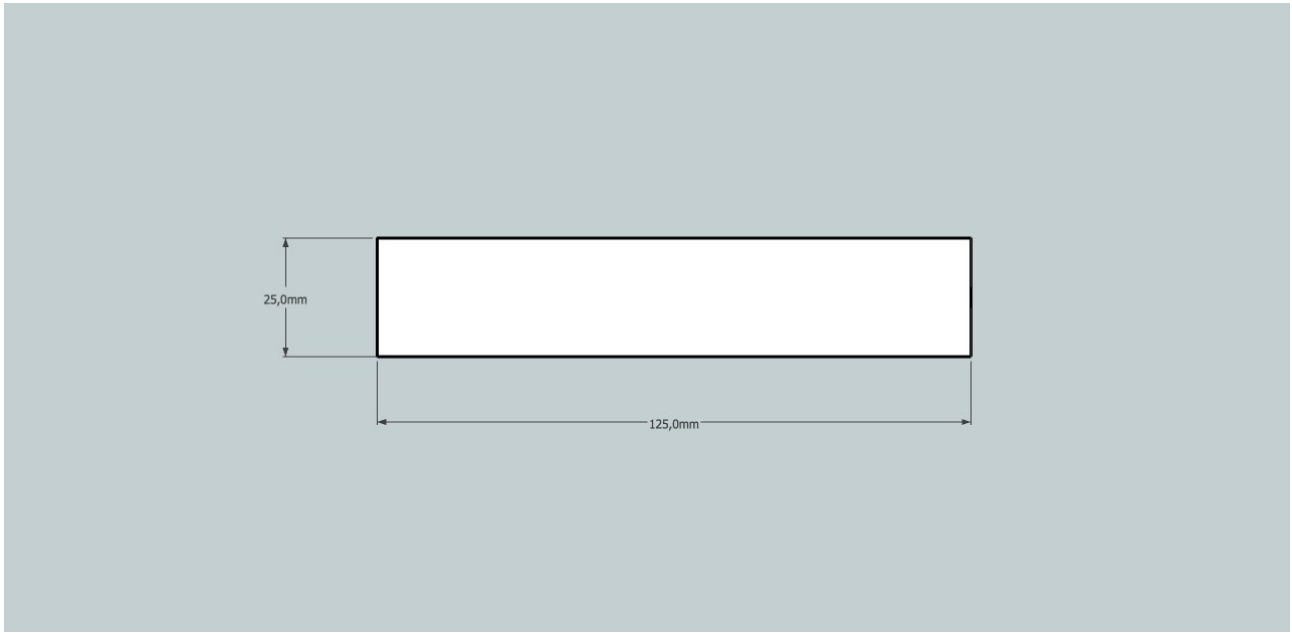


Ilustración 14: Vista posterior del cuerpo principal del dispositivo.

Vista inferior del cuerpo principal del dispositivo.

La parte inferior del cuerpo incluye los taladros utilizados para fijar las distintas piezas mediante cuatro escuadras metálicas planas. Las parejas de orificios que se encuentran el la zona central de la pieza frontal y trasera sirven para fijar el soporte de la rueda trasera del dispositivo.

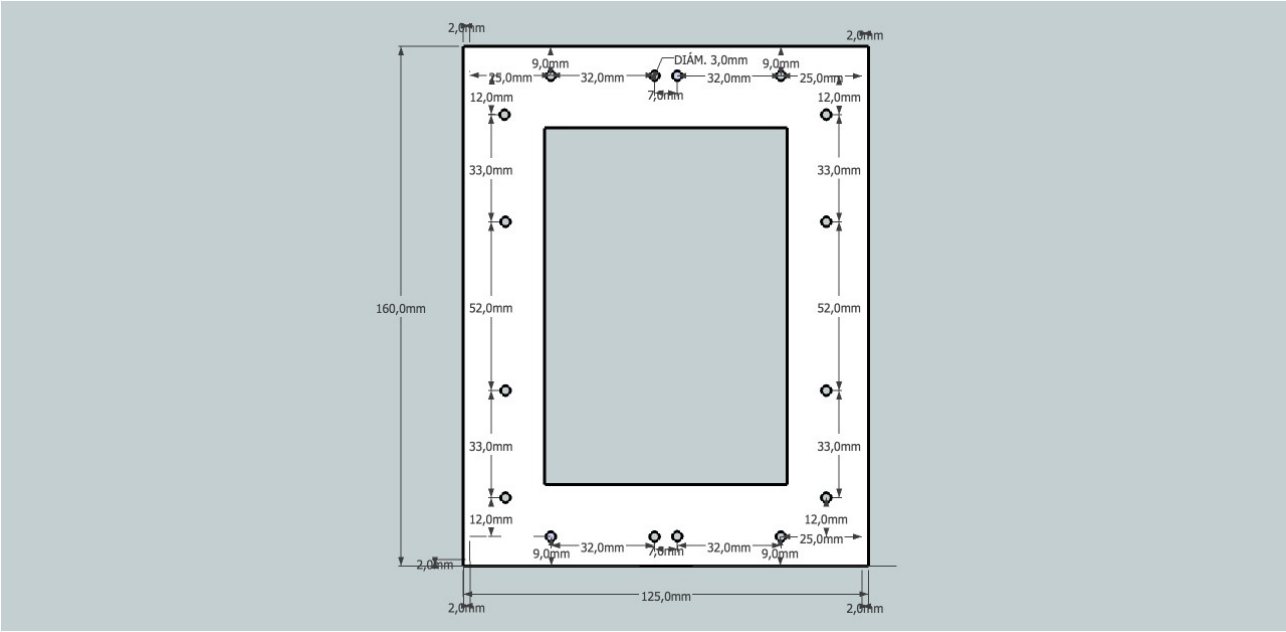


Ilustración 15: Vista inferior del cuerpo principal del dispositivo.

Los primeros taladros situados en las franjas verticales a cada lado se utilizan para fijar los soportes de los servos mediante escuadras metálicas.

Vistas laterales del dispositivo.

Los laterales del dispositivo no alojan ningún sensor ni actuador.

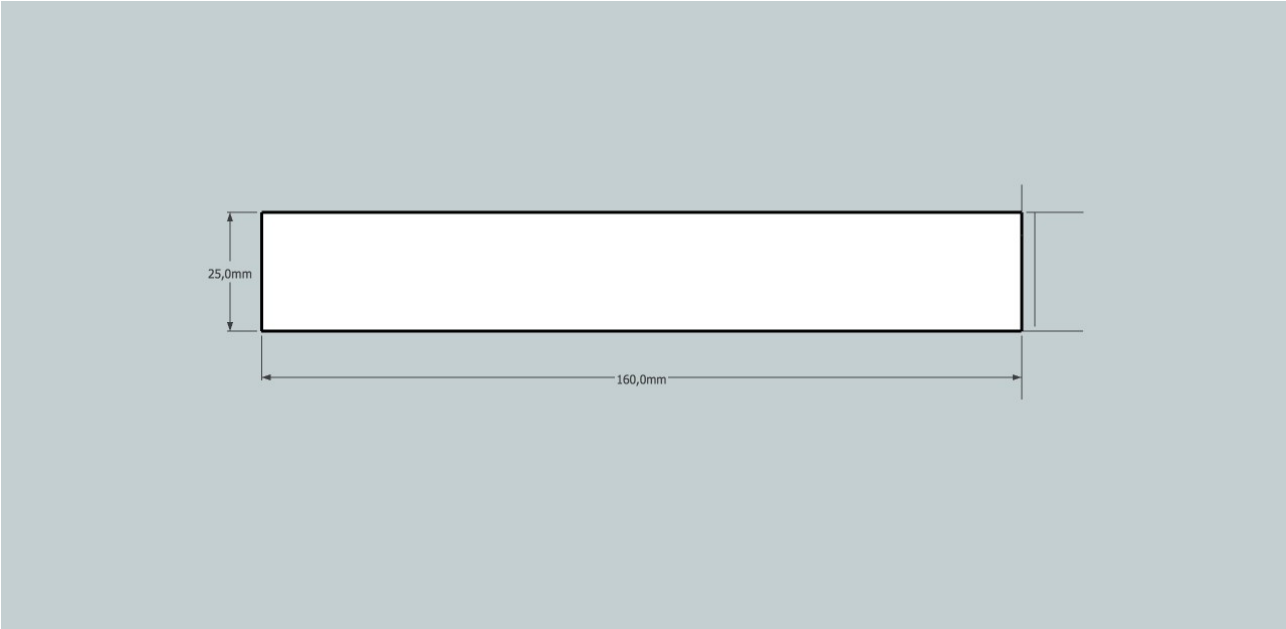


Ilustración 16: Vista izquierda del dispositivo.

Ilustración 17: Vista derecha del cuerpo principal del dispositivo.

Vista en perspectiva del cuerpo principal del dispositivo.

Finalmente, la vista en perspectiva permite apreciar el lugar sobre el que se aloja la lámina de PVC multicell y que sirve para soportar los circuitos y las baterías del robot.

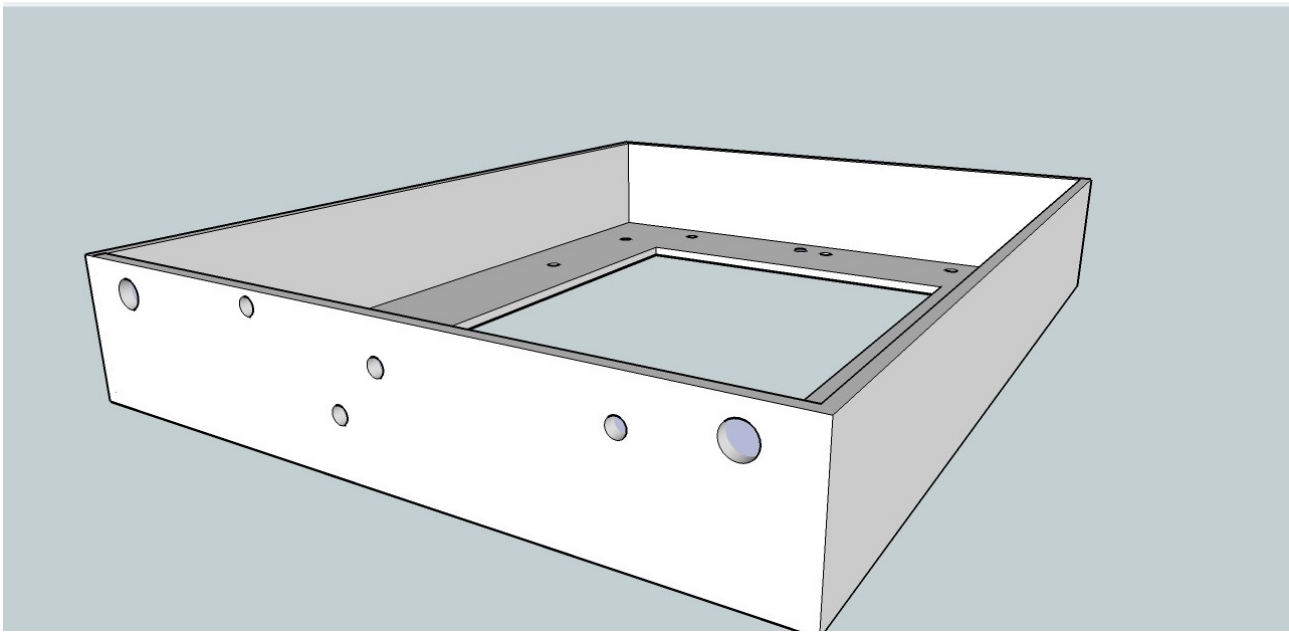


Ilustración 18: Vista en perspectiva del cuerpo principal del dispositivo.

Soporte de la rueda trasera.

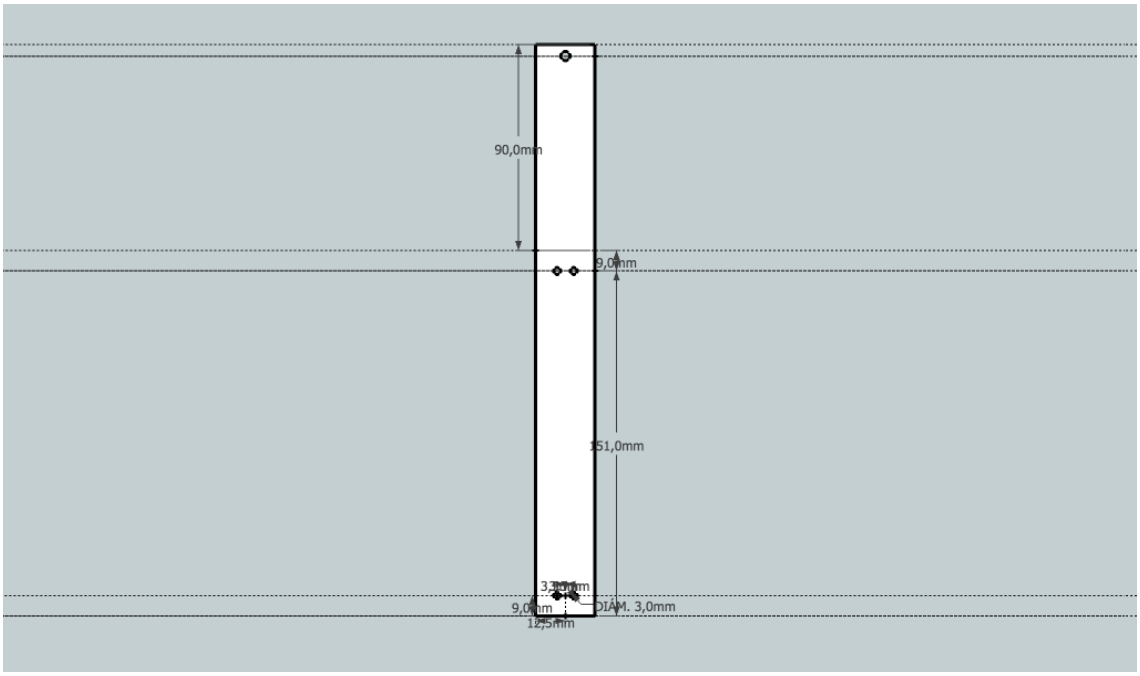


Ilustración 19: Soporte del eje trasero.

A fin de mantener la rueda trasera en su posición correcta y permitir su libre giro se optó por utilizar una lámina de aluminio con un espesor de 2 mm y una anchura de 25 mm. convenientemente taladrada para permitir su unión con el cuerpo principal del dispositivo.

El eje vertical trasero se fabricó a partir de una varilla cilíndrica de aluminio de 6 mm de radio y 45 mm. de largo.

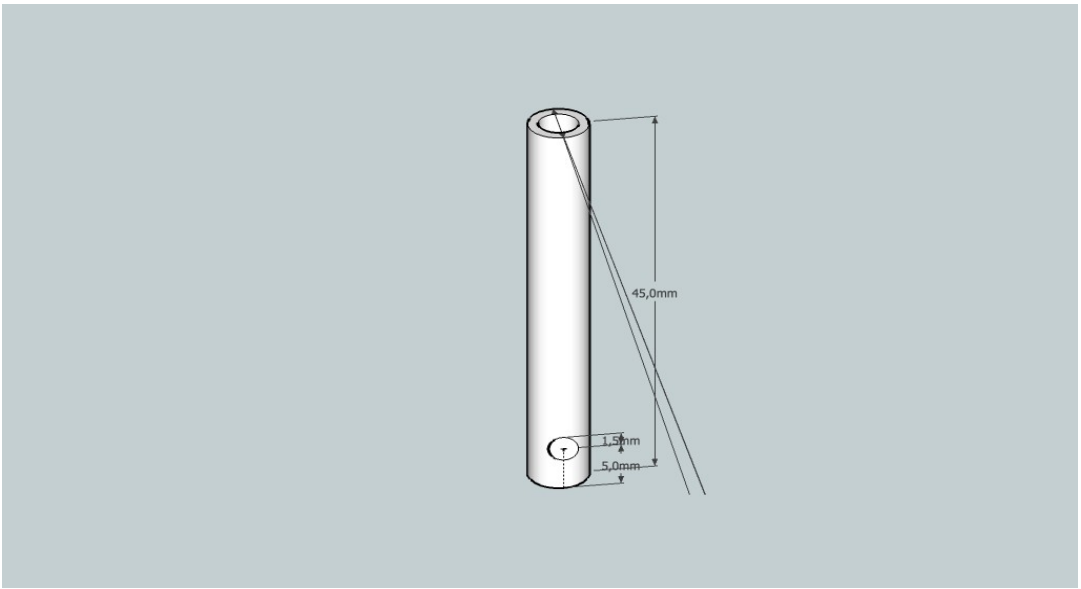


Ilustración 20: Eje vertical trasero.

Soporte para los servos.

Para alojar convenientemente los servos y mantener alineado el sensor óptico de reflexión con el encoder se decidió fabricar una pieza a medida en aluminio. Los orificios superiores sirven para permitir el paso de las escuadras metálicas que conectan estas piezas con el cuerpo principal del robot.

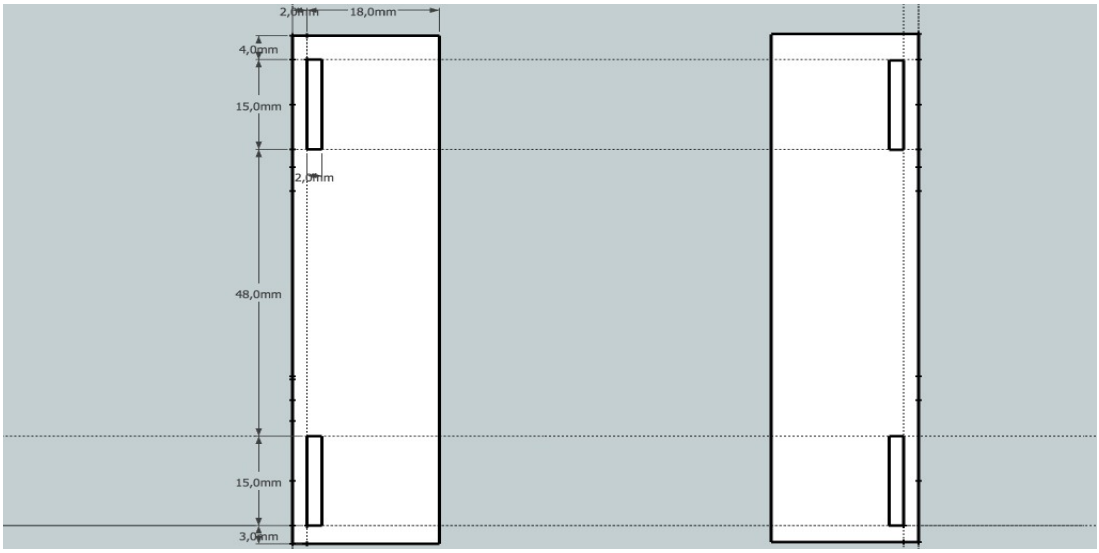


Ilustración 21: Planta de los soportes para los servos.

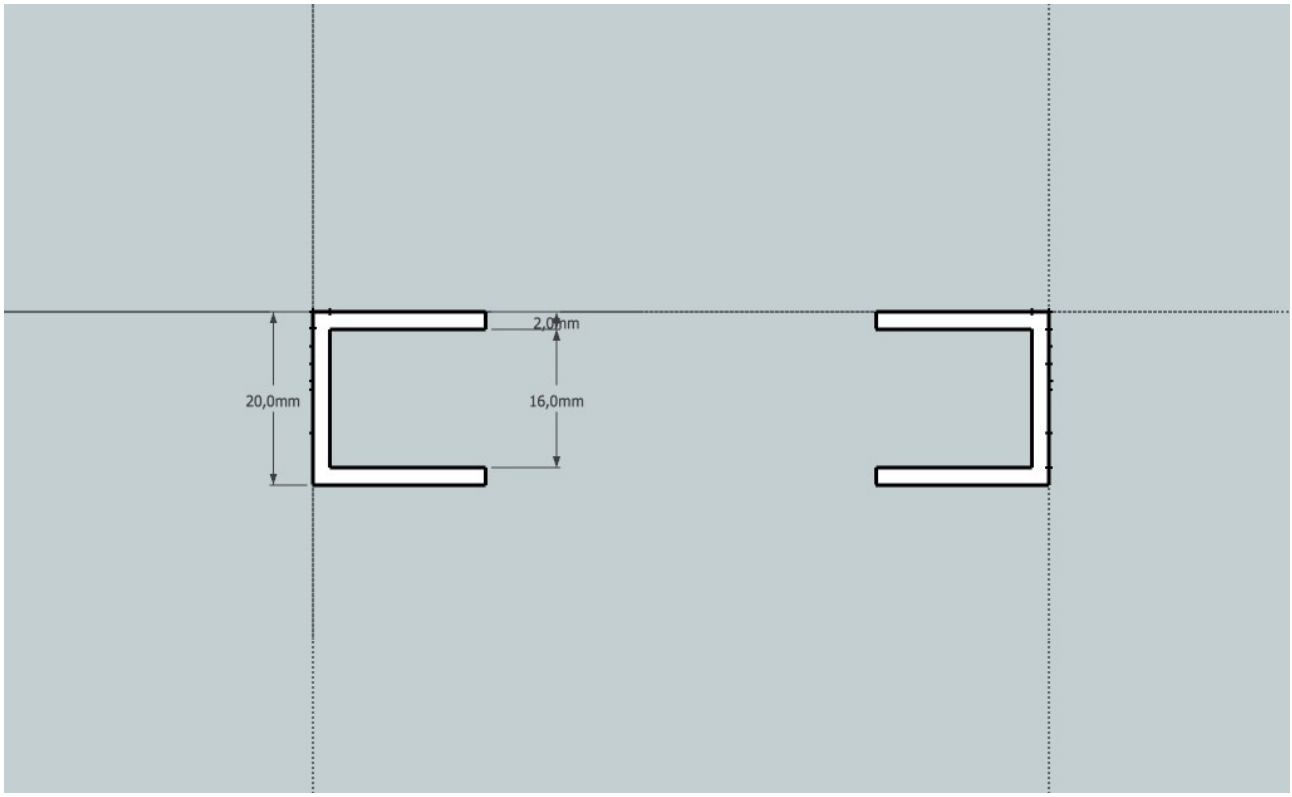


Ilustración 22: Vista frontal de los soportes para los servos.

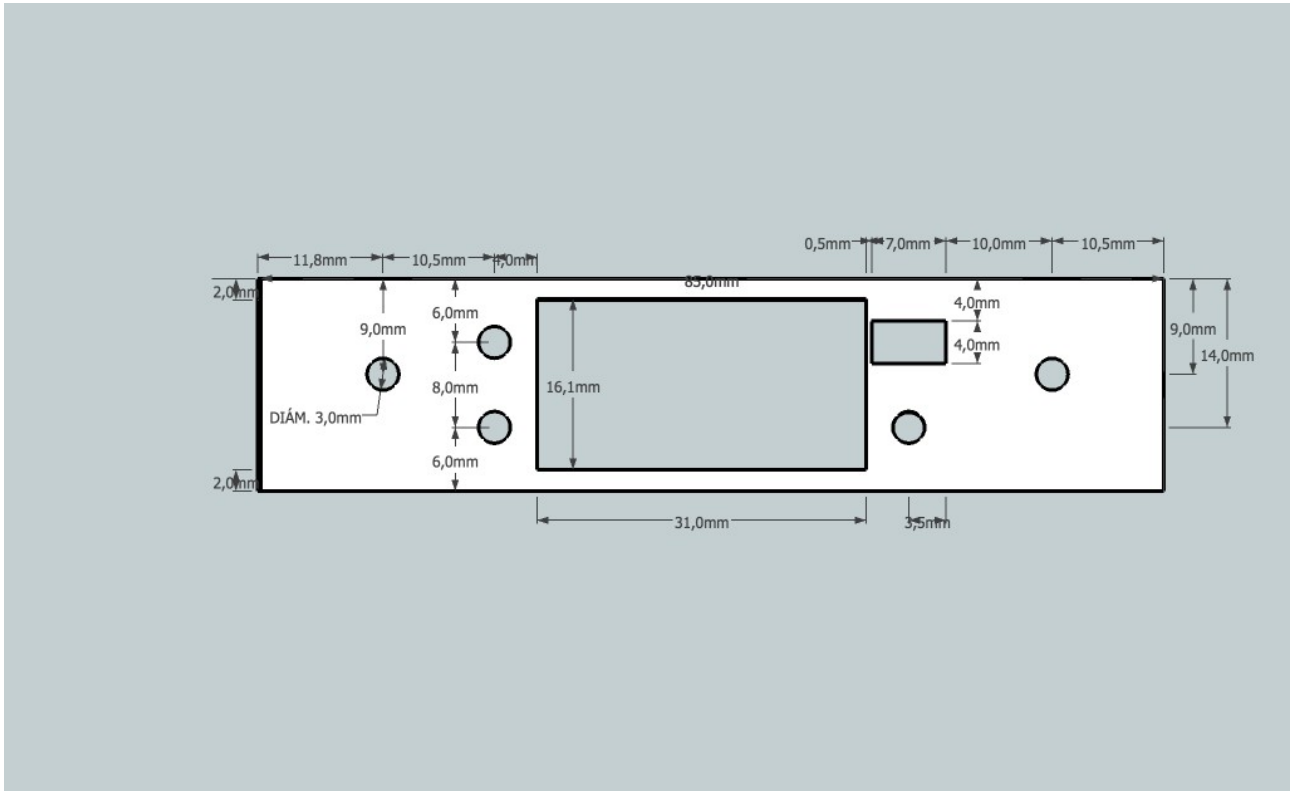


Ilustración 23: Vista lateral de los soportes para los servos. El orificio rectangular aloja los sensores de reflexión.

Integración final.

La siguiente imagen muestra la disposición de los distintos componentes para facilitar la visualización del conjunto.

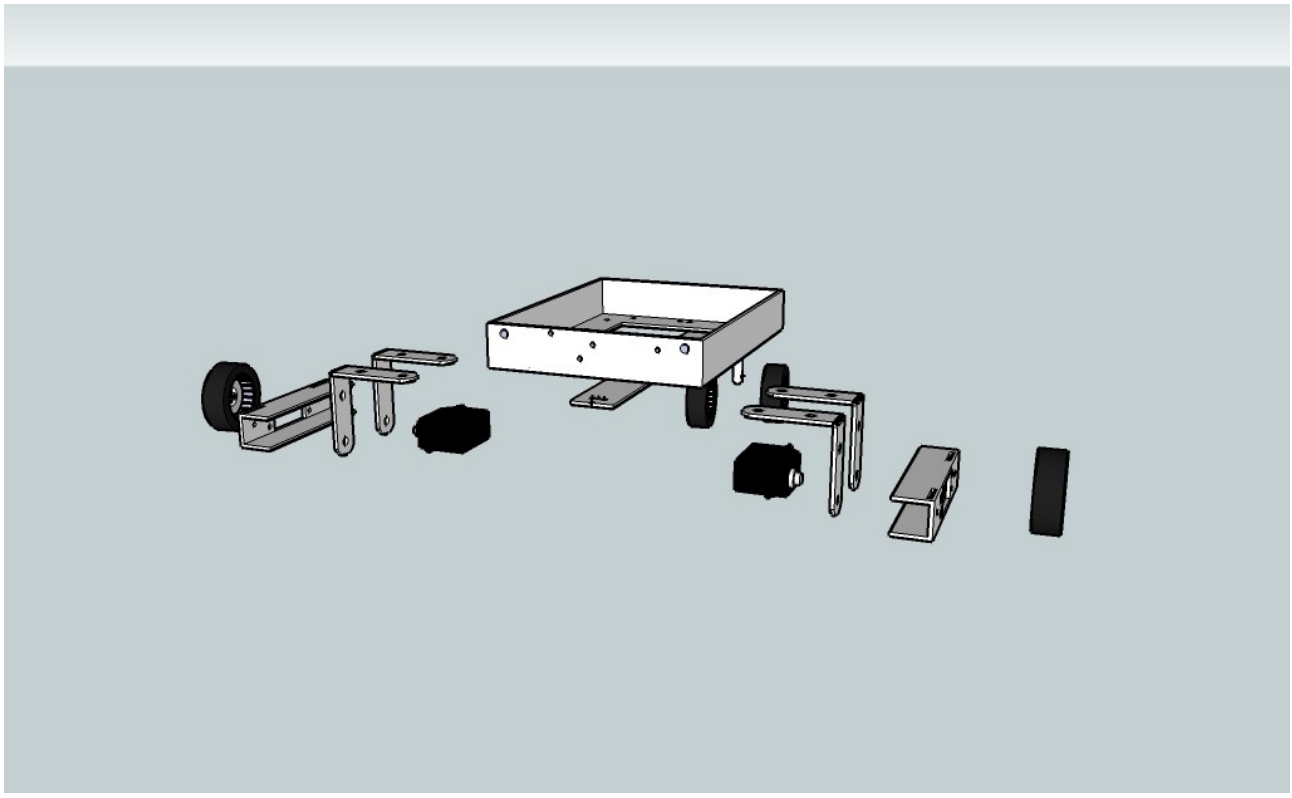


Ilustración 24: Imagen de los componentes desensamblados.

Finalmente se muestra la apariencia real del dispositivo una vez ensamblado.

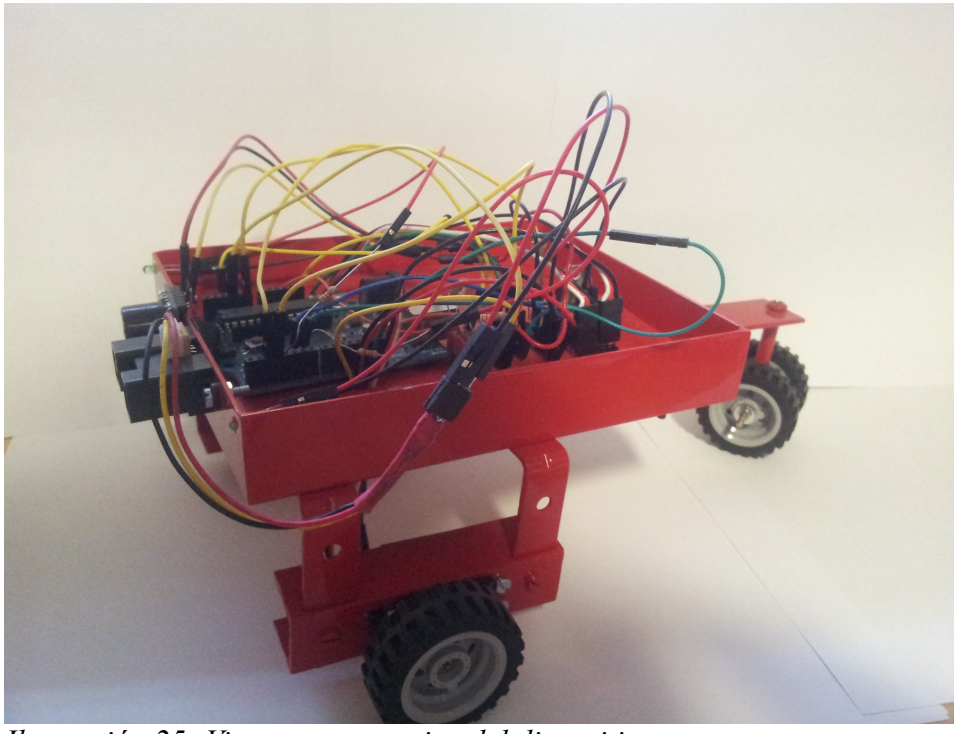


Ilustración 25: Vista en perspectiva del dispositivo

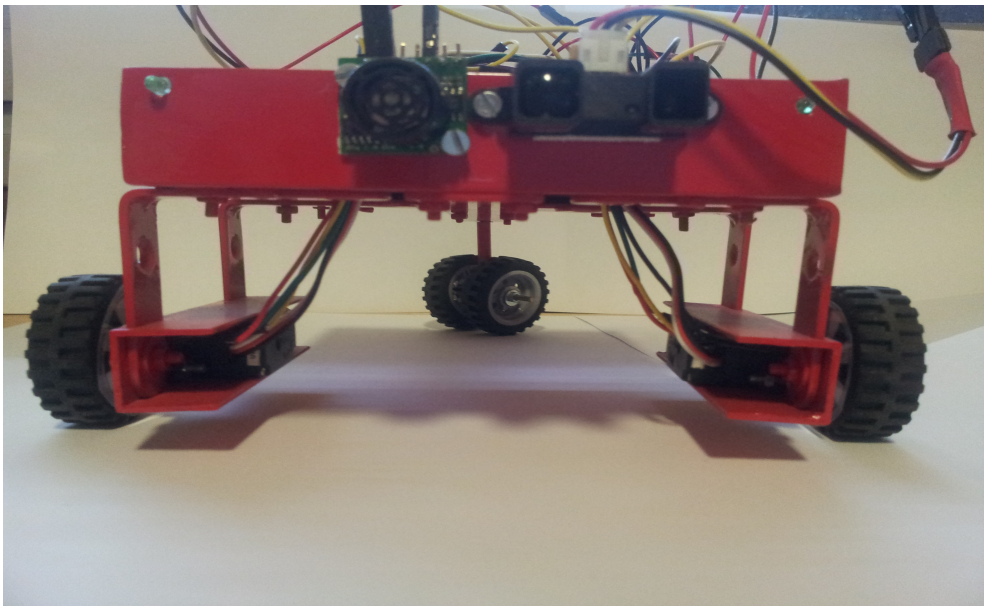


Ilustración 26: Vista frontal del dispositivo

Diseño de la circuitería de soporte.

Para garantizar el adecuado funcionamiento de todos los sensores y actuadores presentes en el dispositivo final deben proveerse una serie de circuitos de estabilización de tensiones así como circuitos de polarización que garanticen que las tensiones y corrientes son las adecuadas.

El siguiente es el diseño del circuito final con todos sus componentes a excepción de la placa WiFly, dado que esta se conecta directamente sobre el Arduino UNO sin requerir circuitería adicional.

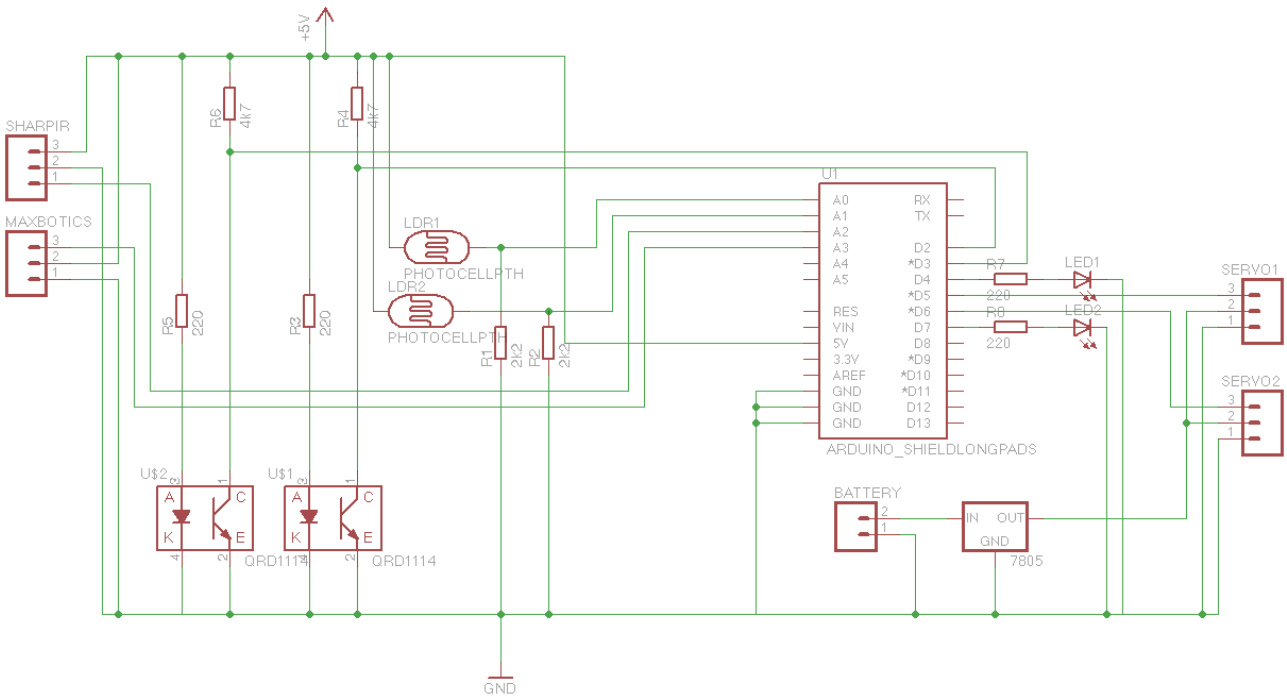


Ilustración 27: Circuito de control de sensores y actuadores.

Para garantizar la interoperabilidad con otras placas de expansión del mercado se optó por diseñar un Shield compatible con Arduino. Sin embargo, el prototipo final se ha montado sobre protoboard por problemas relativos a la fabricación final de la placa de circuito impreso.

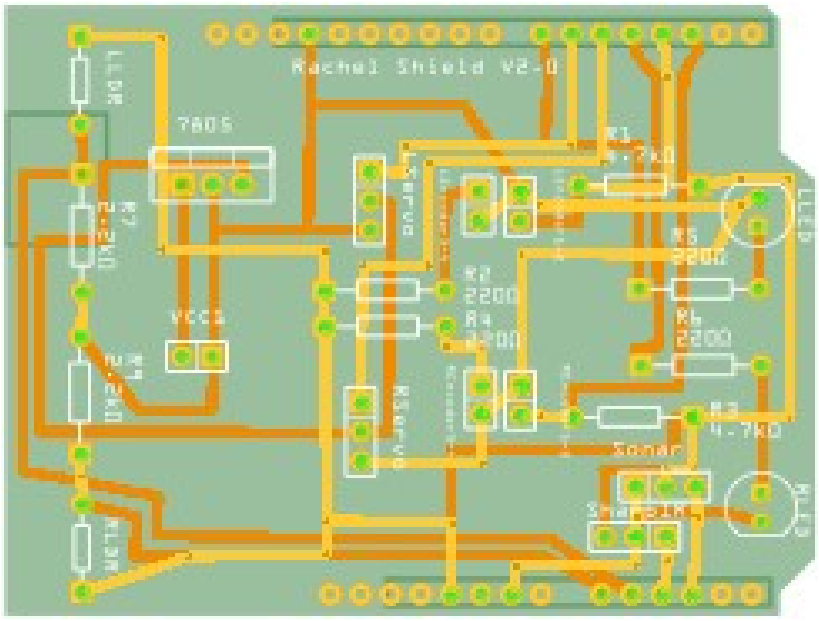


Ilustración 28: Diseño de la placa de circuito impreso.

En cualquier caso se entregan los fotolitos necesarios para la fabricación de la placa de circuito impreso correspondiente.

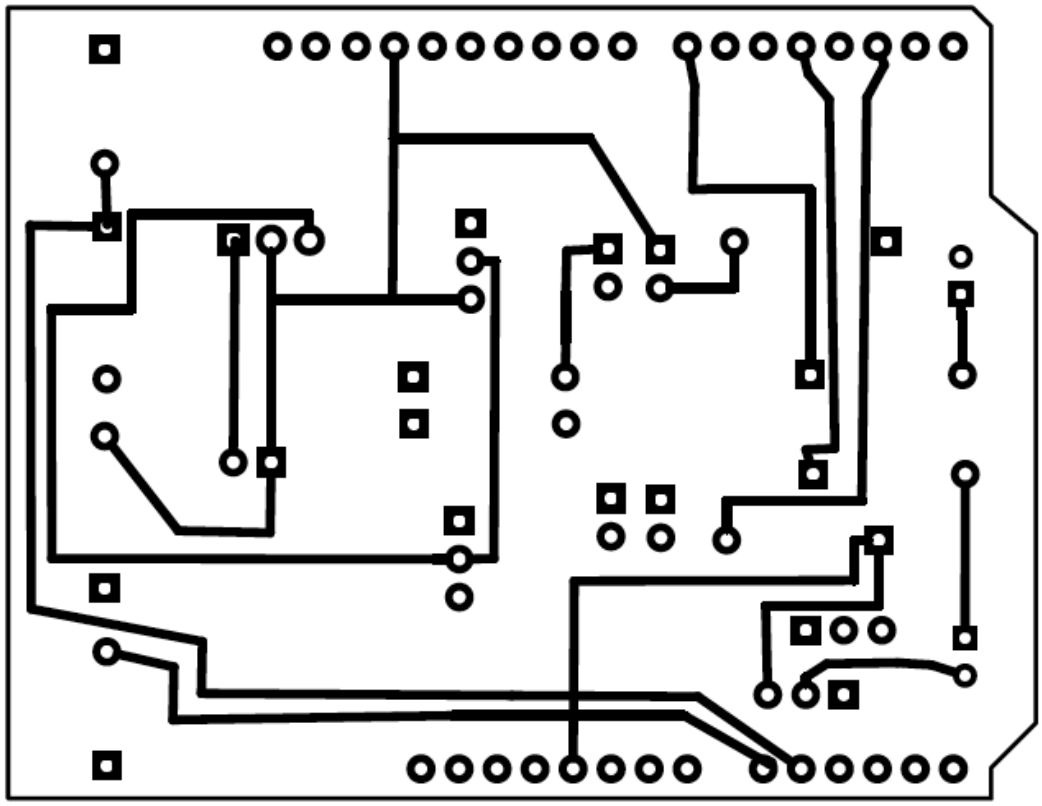


Ilustración 29: Fotolito inferior de la placa de circuito impreso.

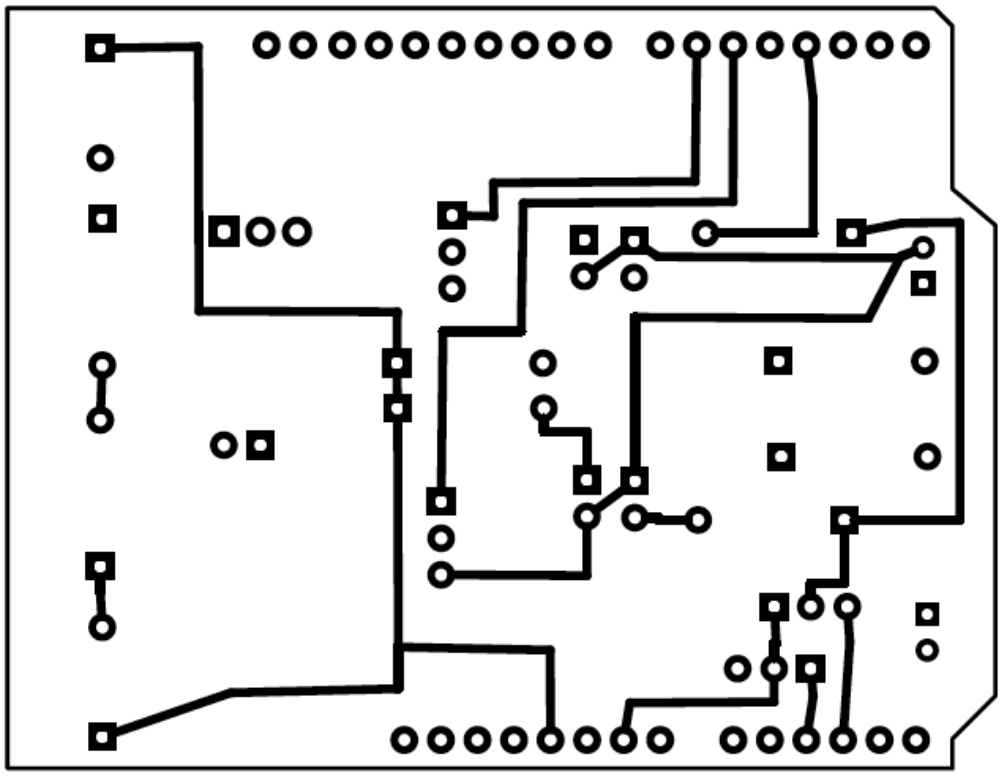


Ilustración 30: Fotolito superior de la placa de circuito impreso.

Librería Replicant.

La librería Replicant habilita a un dispositivo para interpretar los mensajes contemplados en la especificación de la interfaz y para responder adecuadamente a los mismos.

Objetivos de diseño.

Implementar todas las funciones presentes en la especificación Replicant.

Facilitar la distribución de la misma haciendo uso, en la medida de lo posible, de los recursos básicos disponibles en el kit de desarrollo estándar de Arduino.

Adaptarse a la última versión de Arduino, que en este momento es la 1.0.

Promover una interfaz altamente desacoplada que permita la utilización de todas las funciones presentes en la librería por parte de cualquier dispositivo.

Interfaz pública de la librería.

La librería Replicant se estructura alrededor de dos ficheros fuente, concretamente replicant.cpp y replicant.h.

En este último se exponen los diversos métodos que componen la librería así como la definición de las distintas clases.

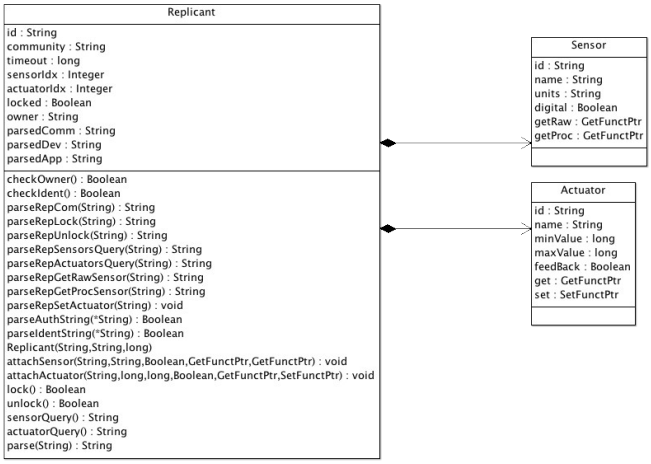


Ilustración 31: Diagrama de clases de la librería Replicant para Arduino

La interacción entre cualquier dispositivo y la librería se lleva a cabo haciendo uso de tres métodos.

El primero de ellos es attachSensor, que se define como sigue:

void attachSensor(String name, String units, bool digital, GetFunctPtr getRaw, GetFunctPtr getProc);

Este método indica a la librería que se ha conectado un nuevo sensor al dispositivo, los argumentos definen su nombre, las unidades en las que toma medidas, si se trata de un sensor que sólo adquiere valores 0 o 1 y por último la función con la que se obtienen los resultados sin procesar y procesados respectivamente. Estos dos últimos parámetros son los que garantizan el bajo acoplamiento con respecto al resto de código del dispositivo. Se trata de dos punteros a función que permiten que la librería sea capaz de llamar en tiempo de ejecución a código nativo del dispositivo sin tener que alterar sensiblemente la estructura anterior a la introducción de Replicant.

El segundo método de uso común es attachActuator, que se define de manera similar al anterior:

void attachActuator(String name, long minValue, long maxValue, bool feedBack, GetFunctPtr g, SetFunctPtr s);

Los argumentos definen el nombre, el valor mínimo que puede enviarse al actuador, el máximo, si se trata de un actuador de bucle cerrado y, de nuevo, dos punteros a funciones que permiten obtener el valor de retorno en el caso de los actuadores de bucle cerrado y establecer el valor que quiere enviarse al actuador.

El último método público es parse, destinado a hacer el análisis léxico de los comandos que recibe por el dispositivo. Su definición es:

String parse(String msg);

Ejemplo de utilización de la librería.

El código mostrado a continuación es parte de la implementación final de Rachel. El ejemplo no es completamente funcional por una cuestión de claridad.

Se señalan en negrita las lineas que deben añadirse para convertir un dispositivo cualquiera en otro con capacidad de autodescribirse y ceder su propio control como se establece en la definición de la interfaz Replicant.

```
#include <Servo.h>

#include "WiFly.h" // We use this for the preinstantiated SpiSerial object.
#include "Replicant.h"



#define RLED_PIN 7
#define LLED_PIN 4
#define RSERVO_PIN 5
#define LSERVO_PIN 6
#define RLDR_PIN A0
#define LLDR_PIN A1
#define SHARP_PIN A2
#define MAXBOTICS_PIN A3
#define RWHEELSENSOR_PIN 2
#define LWHEELSENSOR_PIN 3

int ldr1;
int ldr2;
volatile int rCont;
volatile int lCont;
Servo rServo;
Servo lServo;
long leftLED;
long rightLED;

Replicant r = Replicant("Rachel", "Home", 3000);

void setup() {

 ldr1 = 0;
 ldr2 = 0;
 rCont = 0;
 lCont = 0;
 leftLED = 0;
 rightLED = 0;

 Serial.begin(9600);
```

```
 Serial.println("Initializing Rachel: ");
 Serial.println("Attempting to connect to SPI UART...");
 SpiSerial.begin();
 Serial.println("Connected to SPI UART.");




 pinMode(RSERVO_PIN, OUTPUT);
 rServo.attach(RSERVO_PIN);
 r.attachActuator("Right Servo", 0, 180.0, false, 0, setRightServo);

 pinMode(LSERVO_PIN, OUTPUT);
 lServo.attach(LSERVO_PIN);
 r.attachActuator("Left Servo", 0, 180.0, false, 0, setLeftServo);

 pinMode(RWHEELSENSOR_PIN, INPUT);
 r.attachSensor("Right wheel sensor", "cm", false, getRightWheelRaw, getRightWheelProccessed);

 pinMode(LWHEELSENSOR_PIN, INPUT);
 r.attachSensor("Left wheel sensor", "cm", false, getLeftWheelRaw, getLeftWheelProccessed);

 attachInterrupt(0, rightEncoder, CHANGE);
 attachInterrupt(1, leftEncoder, CHANGE);
}
```

La primera linea instancia un nuevo replicante que e denomina Rachel, pertenece a la comunidad Home y tiene un tiempo máximo de inactividad de 3000 segundos.

Las siguientes líneas registran los sensores y actuadores de que dispone el dispositivo, en este caso, un robot móvil.

En el bucle de control principal podemos ver cómo se inserta toda la funcionalidad de la librería en apenas una linea que también se señala en negrita.

```
void loop() {

 String aux = "";
 String out = "";
 char c;
 while(SpiSerial.available() > 0) {
  c = (char) SpiSerial.read();
  delay(10);
  aux += c;
  if (c == '#') {
    break;
  }
 }
```

```
 if (!aux.equals("")) {
   Serial.println(aux);
   if (c == '#') {
     out = r.parse(aux);
     SpiSerial.println(out);
   }
 }
}
```

Dificultades encontradas.

Durante el desarrollo del dispositivo se produjo el cambio a la versión 1.0 del entorno de desarrollo de Arduino.

Esta última actualización introdujo cambios incompatibles con las versiones anteriores de la plataforma.

Para poder garantizar la continuidad futura del proyecto se optó por desarrollar teniendo como plataforma de desarrollo objetivo la última versión.

Esta decisión provocó una de las mayores dificultades del proyecto. Las librerías de soporte del módulo WiFly Shield de Sparkfun están diseñadas para compilarse contra las versiones anteriores de Arduino, pero no para la versión 1.0. Tras contactar con el fabricante y hacer uso de sus foros de soporte se decidió acometer la reescritura de las librerías para poder cumplir con los plazos de entrega de este proyecto.

Finalmente publiqué la que, hasta la fecha, es la única versión adaptada a Arduino 1.0 de la librería de soporte para Wifly Shield. Para poder cumplir con los plazos impuestos solamente se retocaron aquellas partes de la librería que nos permitirían configurar la interfaz 802.11b de acuerdo a nuestras necesidades.

En la bibliografía se aportan enlaces a la versión de la librería adaptada para este proyecto.

Aplicación móvil para el control de dispositivos Replicant

Alcance de la aplicación

Objetivos.

El desarrollo de esta aplicación móvil pretende demostrar la viabilidad de la interfaz Replicant mediante una prueba de concepto real capaz de interactuar con dispositivos con la habilidad de interpretar las distintas operaciones especificadas. Para ello establecemos como objetivos:

Ejecutarse en cualquier dispositivo con capacidad de conexión a redes de datos y compatible con la especificación Android 2.2 y posteriores.

Habilitar mecanismos para la gestión de comunidades de dispositivos.

Implementar un mecanismo de descubrimiento de dispositivos Replicant visibles a través del uso de una conexión de datos o una conexión física entre el terminal que ejecuta la aplicación y el propio replicante.

Disponer de medios para la consulta de los sensores y actuadores de los que dispone un dispositivo Replicant.

Dotar al usuario de sistemas para bloquear y utilizar un dispositivo Replicant dado.

Entregables.

En este caso y dada la particular arquitectura de las aplicaciones Android, el proceso de desarrollo se planteará entorno a la implementación de actividades entendidas tal y como se definen en la plataforma Android, es decir, interfaces de usuario finales junto con la implementación de los métodos correspondientes de las capas de modelo de negocio y persistencia.

Siguiendo este enfoque los entregables se estructuran del siguiente modo:

1. Gestión de comunidades de dispositivos: Esta actividad debe permitir que un usuario dé de alta una nueva comunidad y pueda añadir nuevos dispositivos a la misma. Del mismo modo, el usuario debe poder modificar y eliminar comunidades ya existentes.
2. Gestión de dispositivos Replicant: Esta actividad permitirá a los usuarios crear nuevos dispositivos asociándolos a comunidades preexistentes. La aplicación debe ser capaz de determinar si un dispositivo implementa la interfaz Replicant o no e indicarlo de manera clara al usuario. En esta misma interfaz, el usuario debe disponer de medios para bloquear el dispositivo garantizándose así su uso exclusivo.
3. Consulta de la lista de sensores y actuadores de un dispositivo replicant: Dada una lista de dispositivos, debe poderse consultar de qué sensores o actuadores dispone cualquiera de ellos. Esta información debería almacenarse para su consulta aunque el dispositivo no esté disponible.
4. Interacción con sensores y actuadores de dispositivos Replicant: Una vez bloqueado un dispositivo esta actividad permitirá que un usuario utilice cualquiera de los sensores y actuadores del mismo.
5. Mecanismo para el uso de operaciones complejas sobre dispositivos Replicant: Esta interfaz contará con medios para la consulta de las operaciones complejas definidas en un dispositivo replicant y para su posterior ejecución.

6. Gestión de suscripciones a eventos Replicant: Esta actividad permitirá consultar los eventos que publica el dispositivo y señalar aquellos de los que se desea obtener información.
7. Mecanismo de recepción de alertas: Esta última actividad será la encargada de recibir y clasificar las alertas a las que el usuario ha decidido suscribirse.

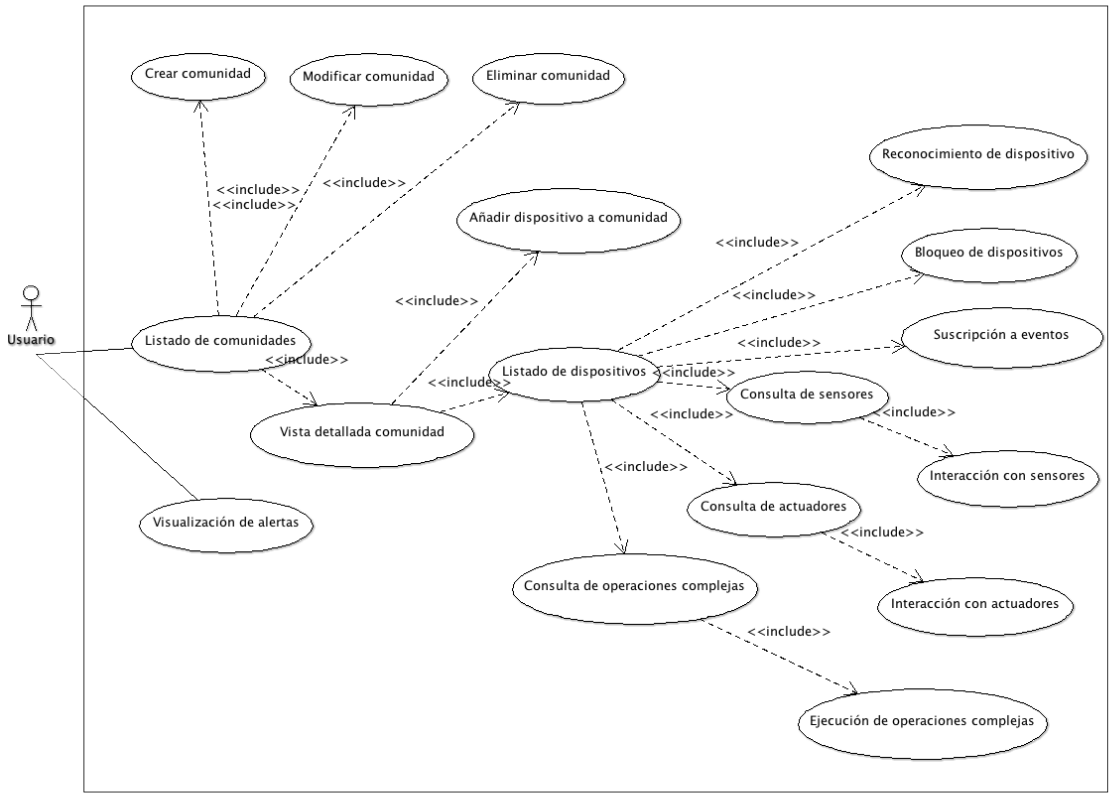
Requisitos.

Toda la aplicación debe someterse a un control de versiones que permita revertir cualquier cambio que provoque resultados no deseados.

La interfaz de usuario de la aplicación debe diseñarse teniendo en cuenta los criterios de buenas prácticas promovidos por la comunidad de desarrolladores Android.

La aplicación debe poder ejecutarse en cualquier terminal Android con capacidad de conexión a redes de datos y compatible con la especificación Android 2.2 o posteriores.

Análisis de casos de uso



Caso de uso 1: Listado de comunidades.

Descripción: Lista las comunidades de dispositivos que se han dado de alta en el sistema.

Actores: Usuario final de la aplicación.

Precondiciones: No aplican.

Flujo normal:

1. El usuario lanza la aplicación y se le presenta la lista de comunidades de dispositivos.
2. El usuario puede seleccionar las opciones de actualización, borrado o adición de nuevas comunidades desde esta interfaz.

Flujo alternativo:

1. Si no existe ninguna comunidad se mostrará un mensaje explicativo.
2. El usuario podrá dar de alta nuevas comunidades desde esta interfaz.

Postcondiciones:

No aplican.

Caso de uso 2: Crear comunidad.

Descripción: Permite la creación de comunidades de dispositivos.

Actores: Usuario final de la aplicación.

Precondiciones: No aplican

Flujo normal:

3. El usuario selecciona la opción añadir comunidad.
4. El usuario provee un nombre para la comunidad.
5. La aplicación comprueba la validez de los datos.

Flujo alternativo:

1. Si existe otra comunidad con el mismo nombre, avisa al actor para que corrija.

Postcondiciones:

Se almacenan los datos en el sistema.

Caso de uso 3: Modificar comunidad.

Descripción: Permite la modificación de comunidades de dispositivos.

Actores: Usuario final de la aplicación.

Precondiciones:

- Debe existir, al menos, una comunidad en el sistema.

Flujo normal:

1. El usuario selecciona la comunidad que desea modificar.
2. El usuario provee un nuevo nombre para la comunidad.
3. La aplicación comprueba la validez de los datos.

Flujo alternativo:

1. Si existe otra comunidad con el mismo nombre, avisa al actor para que corrija.

Postcondiciones:

Se almacenan los datos en el sistema.

Caso de uso 4: Eliminar comunidad.

Descripción: Permite la eliminación de comunidades de dispositivos.

Actores: Usuario final de la aplicación.

Precondiciones:

- Debe existir, al menos, una comunidad en el sistema.

Flujo normal:

1. El usuario selecciona la comunidad que desea eliminar.
2. Se elimina la comunidad del sistema.

Flujo alternativo: No aplica.

Postcondiciones:

Se almacenan los datos en el sistema.

Caso de uso 5: Vista detallada de comunidad.

Descripción: Muestra información sobre los dispositivos que pertenecen a una comunidad dada y sobre su estado.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe seleccionar una comunidad del sistema.

Flujo normal:

1. Se muestra la denominación de la comunidad y un listado de los dispositivos dados de alta en la misma. La interfaz debe permitir conocer el estado de los distintos dispositivos.
2. Se muestra algún mecanismo para asociar nuevos dispositivos a la comunidad.

Flujo alternativo:

- Flujo alternativo 1: No existen dispositivos en la comunidad
    1. Se muestra un mensaje explicativo.

Postcondiciones: No aplica.

Caso de uso 6: Añadir dispositivos a una comunidad.

Descripción: Añade un dispositivo a una comunidad.

Actores: Usuario final de la aplicación.

Precondiciones:

- Debe existir, al menos, una comunidad en el sistema.

Flujo normal:

1. El usuario selecciona la comunidad a la que desea añadir un dispositivo.
2. Introduce el nombre del dispositivo, su dirección y, en su caso, el puerto para la conexión.
3. Se da de alta un nuevo dispositivo en la comunidad.

Flujo alternativo:

- Flujo alternativo 1: El dispositivo ya es parte de la comunidad.
    1. No se añade el dispositivo duplicado a la comunidad.
    2. Se vuelve a mostrar la vista de comunidades.

Postcondiciones: No aplica.

Caso de uso 7: Listado de dispositivos.

Descripción: Lista los dispositivos asociados a una comunidad concreta y permite interactuar con ellos.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado una comunidad del sistema.

Flujo normal:

1. El usuario obtiene una lista de los dispositivos asociados a una comunidad con datos relativos a si han sido reconocidos como replicantes o no.
2. Se habilitan mecanismos para dar de alta, actualizar o eliminar dispositivos de la comunidad.
3. Se dispone de medios para seleccionar un dispositivo concreto del listado.

Flujo alternativo:

- Flujo alternativo 1: No existe ningún dispositivo asociado a la comunidad.

- La aplicación muestra un mensaje explicativo.

Postcondiciones: No aplica.

Caso de uso 8: Reconocimiento de dispositivos.

Descripción: Permite establecer si un dispositivo es compatible con la especificación Replicant o no.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo.
- Debe disponerse de conectividad.

Flujo normal:

1. El usuario selecciona la opción de detectar dispositivos.
2. Se envía un mensaje al dispositivo.
3. Se recibe la respuesta del dispositivo con su identificador dentro de la comunidad.
4. La aplicación verifica que el identificador coincide con el esperado.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.
    1. La aplicación muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se almacena el estado del dispositivo indicando que no es compatible con la interfaz Replicant.
    3. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones:

- Se almacena el estado del dispositivo en el sistema indicando que es compatible con la interfaz Replicant.

Caso de uso 9: Bloqueo de dispositivos.

Descripción: Permite solicitar el uso en exclusividad de un dispositivo Replicant.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe disponerse de conectividad.

Flujo normal:

1. El usuario selecciona la opción de bloquear dispositivos.
2. Se envía un mensaje al dispositivo.
3. Se recibe la respuesta del dispositivo indicando que el bloqueo es efectivo.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.
    1. La aplicación muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se almacena el estado del dispositivo indicando que no está bloqueado.
    3. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 3: El dispositivo responde indicando que la operación de bloqueo no se ha podido llevar a cabo con normalidad.
    1. Se almacena el estado del dispositivo indicando que no está bloqueado.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones:

- Se almacenan los datos en el sistema.

Caso de uso 10: Consulta de sensores.

Descripción: Permite obtener un listado de los sensores conectados a un dispositivo.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe disponerse de conectividad.

Flujo normal:

1. Se envía un mensaje al dispositivo.
2. Se representa el listado de sensores conectados al dispositivo.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.

1. La aplicación muestra un mensaje de error.
2. Se vuelve a mostrar la vista de dispositivo.

- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 3: El dispositivo responde indicando que no tiene sensores conectados.
    1. Se muestra un mensaje explicativo.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones: No aplica.

Caso de uso 11: Consulta de actuadores.

Descripción: Permite obtener un listado de los actuadores conectados a un dispositivo.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe disponerse de conectividad.

Flujo normal:

1. Se envía un mensaje al dispositivo.
2. Se representa el listado de actuadores conectados al dispositivo.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.
    1. La aplicación muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 3: El dispositivo responde indicando que no tiene actuadores conectados.
    1. Se muestra un mensaje explicativo.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones: No aplica.

Caso de uso 12: Consulta de operaciones complejas.

Descripción: Permite obtener un listado de las operaciones complejas definidas para un dispositivo.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe disponerse de conectividad. En caso contrario la aplicación debe mostrar un mensaje de error.

Flujo normal:

1. Se envía un mensaje al dispositivo.
2. Se representa el listado de operaciones complejas definidas en el dispositivo.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.
    1. La aplicación muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 3: El dispositivo responde indicando que no tiene definida ninguna operación compleja.
    1. Se muestra un mensaje explicativo.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones: No aplica.

Caso de uso 13: Interacción con sensores.

Descripción: Permite obtener datos de un sensor concreto.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe seleccionar el sensor del que pretende obtener datos.
- Debe disponerse de conectividad. En caso contrario la aplicación debe mostrar un mensaje de error.

Flujo normal:

1. El usuario selecciona el tipo de dato, normal o procesado, que desea obtener.

2. Se envía un mensaje al dispositivo.
3. Se representa el valor devuelto por el sensor.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.
    1. La aplicación muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones: No aplica.

Caso de uso 14: Interacción con actuadores.

Descripción: Permite interactuar con un actuador concreto.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe seleccionar el actuador con el que pretende interactuar.
- Debe disponerse de conectividad. En caso contrario la aplicación debe mostrar un mensaje de error.

Flujo normal:

1. El usuario selecciona el dato que quiere enviar al dispositivo.
2. Se envía un mensaje al dispositivo.
3. Si el actuador es de bucle cerrado se espera por la respuesta desde el dispositivo.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.
    1. La aplicación muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones: No aplica.

Caso de uso 15: Ejecución de operaciones complejas.

Descripción: Permite ejecutar una acción compleja de las disponibles en un dispositivo.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe seleccionar la operación compleja que quiere ejecutar.
- Debe disponerse de conectividad. En caso contrario la aplicación debe mostrar un mensaje de error.

Flujo normal:

1. El usuario selecciona los argumentos con los que desea ejecutar la operación.
2. Se envía un mensaje al dispositivo.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.
    1. La aplicación muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.
- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones: No aplica.

Caso de uso 16: Suscripción a eventos.

Descripción: Permite suscribir la aplicación a alguno de los eventos exportados por el dispositivo.

Actores: Usuario final de la aplicación.

Precondiciones:

- El usuario debe haber seleccionado un dispositivo compatible con la interfaz Replicant.
- Debe disponerse de conectividad. En caso contrario la aplicación debe mostrar un mensaje de error.

Flujo normal:

1. El usuario selecciona el evento al que quiere suscribirse.
2. Se envía un mensaje al dispositivo.

Flujo alternativo:

- Flujo alternativo 1: El sistema no dispone de conectividad.

1. La aplicación muestra un mensaje de error.
2. Se vuelve a mostrar la vista de dispositivo.

- Flujo alternativo 2: No se recibe respuesta del dispositivo.
    1. Se muestra un mensaje de error.
    2. Se vuelve a mostrar la vista de dispositivo.

Postcondiciones: No aplica.

Arquitectura propuesta.

La plataforma de desarrollo Android promueve activamente el uso de un enfoque arquitectónico basado en el patrón Modelo-Vista-Controlador, también conocido como patrón MVC.

Todas las interfaces gráficas se definirán en ficheros XML que se relacionarán, a posteriori, con clases descendientes de Activity en las que se definirán las operaciones que se llevarán a cabo con los datos introducidos por el usuario.

En el ámbito de la capa de persistencia de datos se propone la implementación de un nivel de abstracción adicional.

Las aplicaciones Android recurren, de manera habitual, a la utilización de bases de datos sqlite 3.0 para hacer persistente la información.

La plataforma Android pone a disposición de sus desarrolladores una clase denominada ContentProvider que permite controlar el acceso concurrente a los datos de manera transparente y evita la labor de mantener manualmente, un único acceso a la base de datos entre las múltiples interfaces de usuario que puedan requerir los datos de manera simultánea.

Además de la comodidad a la hora de desarrollar, y la gestión transparente de la concurrencia, el uso de la clase ContentProvider abstrae totalmente la capa de persistencia del resto de la aplicación y nos permitirá, en un futuro, sustituir la base de datos local del terminal móvil, por servicios alojados en la web que permitan mantener información sobre comunidades de dispositivos replicant a escala global. Bastará con que estos servicios implementen alguna interfaz basada en webservices o alguna tecnología similar.

Modelo de datos.

La capa de persistencia de la aplicación utilizará una base de datos relacional sqlite 3.0.

El modelo de datos consta de siete tablas que representan cada una de las entidades de la aplicación.

Como restricciones adicionales no se permitirá la coexistencia de más de un dispositivo con el mismo nombre en una comunidad.

Del mismo modo, ningún sensor, actuador, operación compleja o evento relacionado con un dispositivo podrá tener el nombre que otro elemento del mismo tipo.

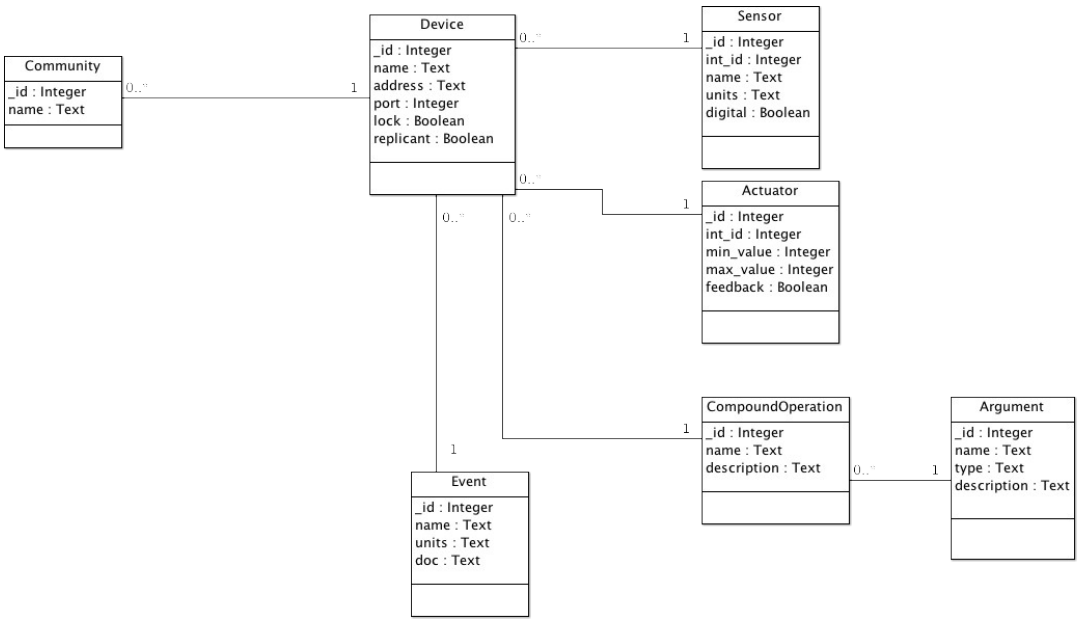


Ilustración 32: Modelo de datos.

Modelo de clases.

La aplicación se estructura en torno a dos paquetes. El paquete principal en el que se almacenan las clases relacionadas con las distintas actividades con las que interactua el usuario final y el paquete "provider" en el que se almacenan todas las clases relacionadas con el componente ContentProvider y con la gestión de la capa de persistencia.

La clase ReplicantData sirve como clase-contrato exportando de manera centralizada los identificadores de los distintos elementos accesibles a través del ContentProvider. El uso de esta clase garantiza una mayor facilidad de mantenimiento y un alto desacoplamiento entre las actividades y la capa de persistencia.

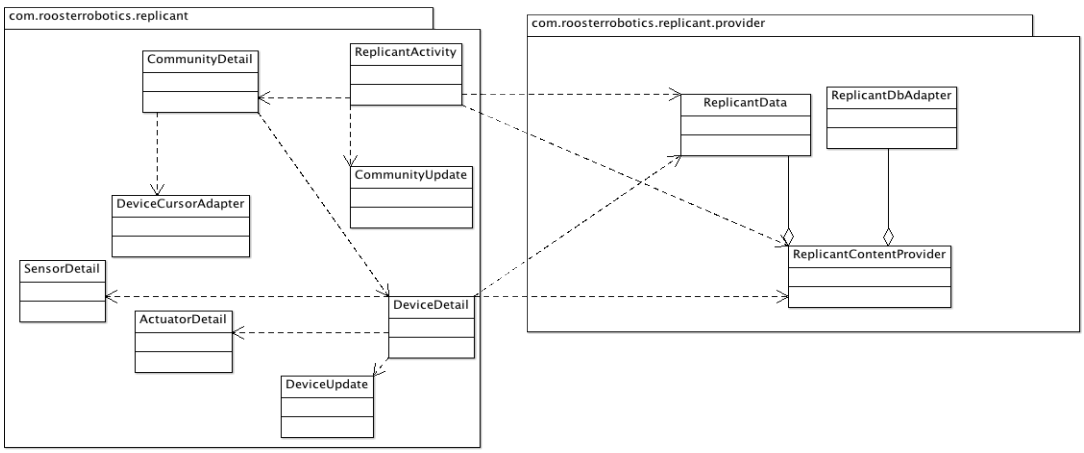


Ilustración 33: Modelo de clases.

Diagrama de navegación.

Ilustración 34: Diagrama de navegación.

La aplicación se ha desarrollado teniendo en cuenta las recomendaciones sobre diseño de interfaces Android.

Se ha primado el uso de componentes que puedan resultar familiares a cualquier usuario habitual de dispositivos móviles.

Como criterio de diseño se ha intentado minimizar el número de interacciones que el usuario final debe realizar para lograr un objetivo final como puede ser bloquear un dispositivo o interactuar con los sensores y actuadores.

Créditos de los recursos gráficos.

La iconografía de la aplicación no ha sido diseñada expresamente para Replicant, sin embargo se han utilizado recursos con licencia libre, a continuación se relacionan las distintas imágenes con su licencia correspondiente y una referencia a su autor original.

Lock open icon	
Icon set	Free tabs color icons
Autor	Kevin Andersson
Licencia	Freeware permitido el uso comercial.
Referencia	http://www.tabsicons.com

Lock icon	
Icon set	Free tabs color icons
Autor	Kevin Andersson
Licencia	Freeware permitido el uso comercial.
Referencia	http://www.tabsicons.com

Unicorn final	
Icon set	Blade Runner icons
Autor	Corwins
Licencia	Creative Commons 3.0 unported permitido el uso comercial.
Referencia	www.corwins.deviantart.com

Home	
Icon set	Pretty Office 4
Autor	CustomIconDesign
Licencia	Gratuito para uso no comercial.
Referencia	http://www.iconspedia.com/pack/pretty-office-4-2793/30.html

Wheels	
Icon set	Mini Icon Set Part 1 Icon Pack
Autor	Customicondesign
Licencia	Gratuito para uso personal.
Referencia	www.customicondesign.com

Satellite	
Icon set	GIS GPS Map Icon Pack
Autor	IconsLand
Licencia	Gratuito para uso no comercial
Referencia	www.icons-land.com

Conclusiones.

El desarrollo de esta interfaz, del dispositivo y de la aplicación móvil de control se ha dilatado durante dos semestres.

Las principales dificultades que han surgido han sido las relativas a la actualización de las librerías de soporte del módulo de comunicación inalámbrica del dispositivo.

Este módulo es un elemento crucial para la comunicación entre el terminal Android y el robot móvil. No existen alternativas comerciales disponibles a un coste razonable, por lo que se decidió dedicar tiempo a acometer la tarea de actualizar las librerías del fabricante, que ha día de hoy, no disponen de otra versión utilizable con el nuevo entorno de desarrollo de Arduino.

En cuanto al desarrollo del dispositivo, las dificultades surgieron alrededor de la fabricación de las placas de control. Finalmente, a pesar de disponer de los diseños de las mismas para integrar todos los subsistemas entorno a una placa de circuito impreso, se desechó este enfoque y se implementó la lógica de control sobre una placa de prototipos.

En el análisis inicial del proyecto, concretamente en el apartado de selección de la plataforma ya hacía mención al carácter espartano de los entornos integrados de desarrollo para microcontroladores. Sin embargo, lo que no llegué a valorar en ningún caso fue la posibilidad de que los datos con respecto a la memoria disponible en el dispositivo que ofrecen los compiladores pudieran ser inexactos.

Durante la programación de la librería para Arduino, guiado por los datos que me ofrecía el entorno de desarrollo, comencé a experimentar gran cantidad de errores inexplicables en primera instancia. Tras varias horas de análisis la conclusión fue que el gran número de cadenas de texto almacenadas en memoria estaban desbordando la RAM de Arduino, a pesar de que según el compilador aún disponía de más de la mitad de la memoria. La solución consistió en migrar todas las cadenas de texto constantes a la memoria Flash.

A raíz de todos estos imprevistos que no se tuvieron en consideración en el análisis inicial de riesgos, la implementación final de la interfaz no ha incluido ninguna implementación de la gestión de operaciones compuestas, de la funcionalidad de notificación de eventos ni del envío de órdenes a actuadores de bucle cerrado.

La reflexión final sobre la implementación de la interfaz Replicant es que ésta es viable, aunque habría que asumir un compromiso entre el objetivo de ofrecer facilidades al desarrollador a la hora de trazar la ejecución de las distintas órdenes y la eficiencia del uso de la memoria de los microcontroladores.

Como líneas futuras de desarrollo considero que deberían acometerse las siguientes:

Debe diseñarse una tabla de traducción entre las cadenas de texto propuestas en la interfaz original y una representación binaria más eficiente en cuanto al uso del canal y a su almacenamiento en memoria.

Sería positivo disponer de un analizador léxico y sintáctico más eficiente, así como incluir en la interfaz los distintos mensajes de error fijando un formato de los mismos.

Habida cuenta de los objetivos iniciales del proyecto, no parece muy coherente la idea de almacenar en una base de datos local dentro del terminal Android todas las referencias a las comunidades y los dispositivos que las conforman.

Una linea futura de desarrollo podría ser la inclusión de un servicio global de gestión de comunidades “en la nube”. La idea es poder disponer de los datos relativos a las comunidades y poder compartirlos entre terminales de una misma organización o empresa evitando la necesidad de

tener que darlos de alta manualmente. La adaptación de la aplicación actual a este nuevo paradigma consistiría en sustituir el backend de la clase ReplicantContentProvider por otra que hiciera uso de una interfaz SOAP o REST.

Por último y en relación con el punto anterior, creo que habría que asociar interfaces de usuario específicas a sensores y actuadores concretos.

Si dispusiésemos del servicio centralizado expuesto anteriormente, podríamos contar con un catálogo de sensores y actuadores relacionados con los elementos de la interfaz gráfica con los que interactuarían de manera preferente. Por ejemplo, en lugar de obtener un valor en centímetros de un sensor medidor de distancias, podríamos representar la información obtenida por ese sensor en una gráfica. En lugar de utilizar un control deslizante para enviar un valor a cualquier actuador, podríamos discriminar entre actuadores que toman valores discretos y actuadores del tipo encendido/apagado y mostrar un control más acorde con la naturaleza del dispositivo.

Esto nos permitiría, incluso, la grabación de macros que, haciendo uso de las operaciones compuestas definidas en los dispositivos y sus notificaciones de eventos, nos habilitase para automatizar tareas realmente complejas.

Referencias Bibliográficas.

Bibliografía relacionada con el desarrollo de la interfaz Replicant.

1. **Hickman, R; Kohler, D; Conley, K; Gerkey, B.** (2011, 11 de Mayo) “Cloud Robotics” Google I/O 2011 [Video Online] <http://www.youtube.com/watch?v=FxXBUp-4800&feature=player_embedded#!>
2. **Martín Cuenca, E.; Angulo Usategui, J; Angulo Martínez, I.** (2003) “Principios, características y aplicaciones generales”. En: Microcontroladores PIC. La clave del diseño (1ª Ed.) Madrid: Thomson Paraninfo.
3. **Jazar, R.** (2007) “Theory of applied robotics. Kinematics, Dynamics and control” ['Teoría sobre robótica aplicada. Cinemática, dinámica y control'] (1ª Ed.) Nueva York: Springer.
4. **Kumar Saha, S.** (2008) “Introducción a la robótica” (1ª Ed.) Noida: McGraw-Hill.
5. **Ollero Baturone, A.** (2001) “Robótica, manipuladores y robots móviles” (1ª Ed.) Barcelona: Marcombo Boixareu Editores.

Bibliografía relacionada con el desarrollo del dispositivo de pruebas.

1. “Página web del proyecto mbed”. Mbed microcontrollers [Artículo en linea] [Fecha de consulta 14 de Febrero de 2012]

   <http://mbed.org/handbook/mbed-Microcontrollers>
2. “Página web del proyecto arduino”. Arduino Uno [Artículo en linea] [Fecha de consulta 14 de Febrero de 2012]

   <http://arduino.cc/en/Main/ArduinoBoardUno>
3. **Martín Cuenca, E.; Angulo Usategui, J; Angulo Martínez, I.** (2003) “Principios, características y aplicaciones generales”. En: Microcontroladores PIC. La clave del diseño (1ª Ed.) Madrid: Thomson Paraninfo.
4. “Página web de Sparkfun”. Hoja de características del sensor infrarrojo de proximidad SHARP GP2Y0A02YK0F. [Documento electrónico] [Fecha de consulta 14 de Febrero de

2012]

<http://www.sparkfun.com/datasheets/Sensors/Infrared/gp2y0a02yk_e.pdf>

5. “Página web de Maxbotics”. Hoja de características del sensor MAXBOTICS MaxSonar EZ 3. [Documento electrónico] [Fecha de consulta 14 de Febrero de de 2012]

   <http://www.maxbotix.com/documents/MB1030_Datasheet.pdf>

6. “Página web de Sparkfun”. Hoja de características del sensor óptico/fototransistor QRD1114 de fairchild semiconductors. [Documento electrónico] [Fecha de consulta 14 de Febrero de 2012]

   <http://www.sparkfun.com/datasheets/BOT/QRD1114.pdf>

7. “Página web de Sparkfun”. Guía de referencia del WiflyShield. [Documento electrónico] [Fecha de consulta 14 de Febrero de 2012]

   <http://www.sparkfun.com/datasheets/Wireless/WiFi/WiFlyGSX-um2.pdf>

8. “Página web de Sparkfun”. Librería Experimental para Arduino del WiflyShield. [Documento electrónico] [Fecha de consulta 14 de Febrero de 2012]

   <http://forum.sparkfun.com/viewtopic.php?p=115626#p115626>

9. Regalado Pacheco, Juan Gregorio (2011, 19 de diciembre) “Blog RoosterTricopter”. Sparkfun Wifly Library Alpha 2 ¿Fixed?. [Documento electrónico] [Fecha de consulta 14 de Febrero de 2012]

   <http://roostertricopter.blogspot.com.es/2011/12/sparkfun-wifly-library-alpha-2-fixed.html>

Bibliografía relacionada con el desarrollo de la aplicación de control móvil.

1. “Página web Android Developers”. Android Developers [Artículo en linea] [Fecha de consulta 3 de Abril de 2012]

   <http://developer.android.com/index.html>

2. “Página web Desarrollo en Android”. Desarrollo en Android [Artículo en linea] [Fecha de consulta 3 de Abril de 2012]

   <http://www.sgoliver.net/blog/?p=1313>

3. **Mednieks, Zigurd; Dornin, Laird; Blake Maike, G.; Nakamura, Masumi.** (2011) En: Programming Android 1st Edition USA: O' Reilly Media.